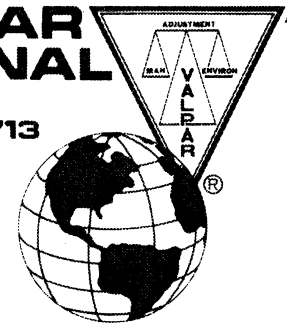
Placement of tabs for valDOS

The tab title included should be cut apart and inserted into the tabs in the following order:

* valDOS CMDS locate before section LXII.
* valDOS system locate before section LXIII.
* File Editor locate before section LXIV.

VALPAR
INTERNATIONAL
3801 E. 34TH STREET
TUCSON, ARIZONA 85713
602-790-7141



valFORTH T.M.

SOFTWARE SYSTEM

for ATARI*

valDOS

*Atari is a trademark of Atari, Inc., a division of Warner Communications.

valFORTH™
SOFTWARE SYSTEM

valDOS

Stephen Maguire

VALPAR INTERNATIONAL

Disclaimer of Warranty
on Computer Programs

All Valpar International computer programs are distributed on an "as is" basis without warranty of any kind. The total risk as to the quality and performance of such programs is with the purchaser. Should the programs prove defective following their purchase, the purchaser and not the manufacturer, distributor, or retailer assumes the entire cost of all necessary servicing or repair.

Valpar International shall have no liability or responsibility to a purchaser, customer, or any other person or entity with respect to any liability, loss, or damage caused directly or indirectly by computer programs sold by Valpar International. This disclaimer includes but is not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of such computer programs.

Defective media (diskettes) will be replaced if diskette(s) is returned to Valpar International within 30 days of date of sale to user.

Defective media (diskettes) which is returned after the 30 day sale date will be replaced upon the receipt by Valpar of a $12.00 Replacement Fee.

valDOS

Table of Contents

Strolling Through valDOS

Until now, one of the major drawbacks of FORTHs for Atari was that you could could either save code on screens or in DOS files, but not both. The demand to have both brought on the development of this two-disk package -- valDOS.

With valDOS you can load FORTH screens just as before, or you can load FORTH source from standard Atari DOS files. But this package allows much more than that. A complete file editor is supplied which can be used to edit FORTH (or assembly, Pascal, etc.) source code and save it in DOS format.

To use this package to its fullest, at least 32K of memory should be available. If you have less than 32K, don't worry, it just means that you can't load all of the commands at once. The package is broken into six major parts:

1) valDOS (approximately 3.6K compiled)
2) valDOS extensions
3) Basic DOS commands (keyboard entry only)
4) DOS command extensions
5) Formatter/copiers
6) File editor

We shall begin this stroll by taking a look at the first five parts. Before starting, duplicate both master disks (we'll be modifying the valDOS II copy). Insert your valDOS I disk in drive one and list screen 170 by typing 170 LIST . Look for the line that says DOS COMMAND EXTENSIONS and load them in. This should take a few minutes. There are so many commands with so many variations on each that only the highlights of the package will be discussed here. Each of the commands is explained in detail in section LXII.

Insert your copy of the valDOS II diskette. Type DIR and press the return key. You should see the following on your display:

```
Files on: valDOS -- disk II

  NAME     EXT    SIZE    SEC    ATTR
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

 ASSM     .4TH      33      4     L
 XBOOT    .4TH      10     37     L
 DOCLISTS.4TH        7     47     L
 FILEIT   .4TH       8     54     L

209 sectors free.
```

The DIR command simply displays the directory of the disk. There are four files on this disk. ASSM.4TH is the standard valFORTH assembler with a correction for the "absolute,Y" instruction which assembled incorrectly when the "absolute" address was in the zero page. XBOOT.4TH is a routine which allows dictionaries greater than 32K to be SAVEd and AUTOed. DOCLISTS.4TH is the routine used to list six screens/page for valFORTH documentation. And finally, FILEIT.4TH is a routine that transfers FORTH screens to DOS files.

If you have a two drive system, DIR 2 will give the directory of the disk in drive two. The DIR command will also accept file specifications. Try the following:

```
DIR  ASSM.4TH
DIR  X*
DIR  ???I*
DIR  MYFILE
```

In these examples, there are the two characters "?" and "*" which need some explanation. Simply put, "?" will match any single letter when compare with a filename in the directory. "*" will match any group of letters to the right from its position to the end of the filename. In the second example above, all files starting with "X" are listed. In the third example, all files whose fifth letter is "I" are listed (the first four can be anything).

For advanced FORTH programmers, the SECtor column of the listing is the first sector of the file. This is a DOS sector (FORTH sectors are offset by -1, i.e., FORTH sectors are numbered starting at zero while Atari has numbered DOS sectors beginning with one; hence, DOS sector 4 is FORTH sector 3). The last column indicates the attributes of the files. All of the files on this disk are locked and therefore have the "L" attribute.

Notice that unlike most FORTH words, DIR expects its arguments on the right. Although valDOS system itself expects all parameters to be passed on the stack, the valDOS user commands all expect their arguments on the right, as is usual with DOS's. The only restriction on this method of input is that no blanks may appear within the command list.

Let's make a copy of one of the files. The COPY command does this nicely for us:

```
COPY  FILE1=FILEIT.4TH
```

This command has the basic format "new=old". The COPY command can also append two or more files together and save this new file into another file:

```
COPY  FILE2=FILEIT.4TH,DOCLISTS.4TH
```

FILE2 should now contain FILEIT.4TH with DOCLISTS.4TH

appended to it. It is possible to string as many files together as desired. We're not quite through yet. If we were to enter:

```
COPY  FILE2=XBOOT.4TH
```

FILE2 would contain an exact copy of XBOOT.4TH. However, if we typed:

```
COPY  FILE2/A=XBOOT.4TH
```

XBOOT.4TH would be appended to whatever was already in FILE2, namely FILEIT.4TH + DOCLISTS.4TH. Of course, also legal is:

```
COPY  FILE2/A=XBOOT.4TH,ASSM.4TH
```

and all of the original files would be contained in FILE2. Unfortunately, FILE2 is FORTH source, but does not have the extension 4TH (it is not needed, but desirable). The RENAME command can rememdy this:

```
RENAME  FILE2.4TH=FILE2
```

Like the COPY command, the format for RENAME is "new=old". For some, typing a long name like RENAME might be a chore. Since we are are in FORTH, we can easily customize:

```
: REN   RENAME ;
```

Now that we have some more files to work with, type

```
DIR  F*
```
and
```
DIR  F*.
```

Notice the difference? In the first example, the remainder of the filename is wild, while in the second example, the files must end with a null extension.

If we want to keep the file FILE1 from being modified in any way, we can lock it using the LOCK command:

```
LOCK  FILE1
DIR
```

The LOCK (and so UNLOCK and KILL) command can take several arguments, separated by commas:

```
LOCK  FILE1,MYFILE,ASSM.4TH,F*.4TH
```

In this case, LOCK would lock FILE1, report MYFILE as being non-existent, lock ASSM.4TH, and then issue a verification prompt before locking any file that matches the specification F*.4TH. Try the following two examples:

```
LOCK  *.4TH
LOCK  *.4TH/N
```

In the first form, a verification prompt is issued for each file before it is locked, while in the second form, the verification message is not displayed. The /N switch stands for "No ask" or "No verify".

Both KILL and UNLOCK have exactly the same argument list as LOCK. Use caution when using the /N switch with the KILL command.

If we want to get a listing of a text file, we use the PRINT command. For example:

```
PRINT  FILE-IT.4TH
```

we get:

File: D1:FILEIT.4TH

```
0001
0002 ( Routine: FILE-IT
0003
0004   The following routine will
0005   transfer a specified range of
0006   FORTH screens to a file on
0007   a DOS formatted disk.
0008
0009   Format:  FILE-IT 1st,last,filename
0010            FILE-IT 10,20,MYFILE
0011
0012   Note that DOS commands tend to be
0013   long because of error checking and
0014   parameter parsing.                  )
0015
0016
0017 : FILE-IT DOS              ( -- )
0018   GETARGS ?WRGARG 44 GETARG ?WRGARG
0019   GETVAL SWAP 44 GETARG ?WRGARG
0020   GETVAL SWAP <ROT OVER - 1+
0021   PAD DUP 1 AND - BUFBOT OVER -
0022   B/BUF / 3 PICK B/SCR * OVER >
0023   IF
0024     CR ." Too many screens, "
0025     B/SCR / . ." max." CR
0026     2DROP 2DROP
0027   ELSE
0028     DROP <ROT 0+S
0029     DO
0030       16 0
0031       DO
0032         I J (LINE)
0033         -TRAILING >R OVER
0034         R CMOVE R> +
0035         155 OVER C! 1+
0036       LOOP
0037     LOOP
0038     PAD DUP 1 AND - SWAP OVER -
0039     FLUSH INSDST ROT DUP (ENTER) 0=
0040     IF FLEXST DSKERR = ?SYSERR ENDIF
0041     (OPEN) ?SYSERR SWAP DROP >R
0042     R (WRITE) ?SYSERR
0043     R (ENDF) ?SYSERR
0044     R> (CLOSE)
0045   ENDIF
0046   CR ;
0047
0048 FORTH
```

The PRINT command has several options. If the switch /N (no line numbers) is present, the file is displayed without line numbers. If an optional starting line number is supplied, printing begins at that line:

```
PRINT  FILEIT.4TH/N,17
```

This will print the file starting with line 17, and with no line numbers. Note that the listing may be aborted at any time by pressing one of the yellow console keys.

The PRINT command is ideal for displaying text files, but is utterly useless for listing a binary or data file. For this reason, the command FDUMP (file DUMP) has been supplied. Let's try FDUMP on FILEIT.4TH:

```
FDUMP FILEIT.4TH

File: D1:FILEIT.4TH

0000  9B 28 20 52 6F 75 74 69  .( Routi
0008  6E 65 3A 20 46 49 4C 45  ne: FILE
0010  2D 49 54 9B 20 9B 20 20  -IT. .
0018  54 68 65 20 66 6F 6C 6C  The foll
0020  ...
```

Like PRINT, FDUMP may be aborted by pressing one of the yellow console keys. If the filename has the "wide" switch /W appended to it, the file is dumped with 16 bytes/line instead of eight as above. This format is more appropriate when sent to a printer. FDUMP always dumps in hexadecimal.

Let's compile a routine from a DOS file. The file DOCLISTS.4TH is the program used to print 6 screens/page for documentation. To load this file, we simply enter:

```
FLOAD  DOCLISTS.4TH
```

and the routine DOCLISTS should now be in the dictionary. There is a nice feature to the FLOAD command:

```
FORGET DOCLISTS
ON ECHO
FLOAD DOCLISTS.4TH
```

Notice that this time the file was echoed to the display as it was being loaded. Like the PRINT command, FLOAD can take an optional starting line so that loading can begin mid-file. Holding down a console key will abort a load once the current definition is compiled.

If you have a source file which is physically contained two or more files, they can be linked together using the FLOAD command. For example, PART1.4TH could end with an

FLOAD PART2.4TH, and part2 could end with FLOAD PART3.4TH, etc. In this way, a multi-part file could be loaded. Although this method works, it should be avoided for several reasons. One reason is that if any of the files are renamed, or if they are moved to a different drive, each of the original files may have to be edited to change the FLOADs. A second and more complicated reason is that for each file FLOADed this way, a separate file buffer needs to be allocated. The default number of file buffers is four, therefore no more than four files can be chained.

The solution to this problem is to create a file which contains nothing but FLOADs:

File: ALLPARTS.4TH

```
0001 FLOAD PART1.4TH  ( load in the ??? routines )
0002 FLOAD PART2.4TH  ( load in the ??? routines )
0003 FLOAD PART3.4TH  ( ... )
```

This method requires exactly two file buffers and allows filenames to be changed easily, if desired.

This was just a brief stroll through the valDOS package. There are many powerful commands left to explore. The following section on command words explains each command in detail. Read through this section carefully and out the commands on a test disk. And then there is the File Editor, with its own set of documentation, in section LXIV.

Have fun.

File: D2:VERIFY.4TH

```
0001
0002 ( Routine: Write w/o verify
0003
0004   The following routine allows write
0005   operations to the disk without
0006   read verification. This speeds up
0007   disk access by many times. Note
0008   that once this routine is loaded,
0009   it may not be forgotten!
0010
0011   Format:  ON  VERIFY
0012            OFF VERIFY
0013                                      )
0014
0015 BASE @ HEX
0016 ASSEMBLER
0017
0018 LABEL -DSK
0019   AD C, 02 C, 03 C, C9 C, 52 C,
0020   D0 C, 05 C, A9 C, 40 C, 4C C,
0021   HERE 4 + ,  A9 C, 80 C, 8D C,
0022   03 C, 03 C, A9 C, 31 C, 8D C,
0023   00 C, 03 C, A9 C, 07 C, 8D C,
0024   06 C, 03 C, A9 C, 80 C,
0025   ( or 00 C, for Percom?) 8D C,
0026   08 C, 03 C, A9 C, 00 C,
0027   ( or 01 C, for Percom?) 8D C,
0028   09 C, 03 C, 20 C, 59 C, E4 C,
0029   60 C,
0030
0031
0032
0033 : VERIFY                          ( f --- )
0034   0# 7 * 50 +
0035   [ ' -DISK 7 + ]
0036   LITERAL C! ;
0037
0038 -DSK ' -DISK 27 + !
0039
0040 BASE !
0041
```

Command Words

Introduction and Conventions

The valFORTH Disk Operating System can be broken into two distinct categories. The first contains the system words which are for use within running programs and are rarely typed directly at the keyboard. The second category contains "command" words which were designed to be executed only at the keyboard. Typical command words are those that list the directory of a disk or delete a file from the disk.

Commands words differ from normal FORTH words in that all necessary arguments are entered following the command word. For example, to remove a file from a directory, we would type:

```
KILL  UNWANTED.FIL
```

instead of the usual FORTH-like:

```
" UNWANTED.FIL"  KILL
```

which will not work as-is. The only restriction placed on this method of input is that absolutely no blanks must appear within the command list since the blank serves to indicate the end of that list. Thus,

```
KILL  FILE1,FILE2,FILE3
```

would properly kill the three files specified while

```
KILL  FILE1, FILE2, FILE3
```

would kill the first file and then abort with an error.

In the command descriptions that follow, any portion of the command format enclosed by the braces "{" and "}" is optional and need not be entered.

Additionally, some of the commands may be aborted by pressing one of the yellow console keys found on the far right of the keyboard. Those commands which have this feature are indicated by the sentence:

"This command is interruptable"

All commands and arguments must be entered in upper case.

CLOSE

Release file buffer and update file.

STATUS:

User memory at PAD is untouched.

COMMAND FORMAT:

CLOSE ⟨filenum⟩

OPERATION:

The CLOSE command flushes the file buffer (different from the FORTH disk buffers), if updated, associated with the specified file number. The disk buffer is then released for a subsequent open (see OPEN). Any future references to the specified file number are ignored until another OPEN command re-assigns it. If a file-number is not specified, or if it is zero, all open files are closed.

EXAMPLES:

CLOSE

Close all open files.

CLOSE 0

Close all open files.

CLOSE 2

Close file number two.

NOTES:

COPY

Transfer the contents of one or more files to another file.

STATUS:

User memory at PAD is untouched.

COMMAND FORMAT:

COPY outfile{/A}=infile1{,infile2{,...}}

OPERATION:

The contents of "infile1" are transferred to "outfile." If it does not already exist, "outfile" is created. If the /A switch is present, the input file is instead appended to the output file. All additional input files are appended to the current output file in the order in which they appear. Single drive users should also see FMOVE.

EXAMPLES:

COPY MYFILE.BAK=MYFILE

Transfer contents of MYFILE to MYFILE.BAK on the default drive unit. (see SETUNIT)

COPY D2:PART1/A=PART2

Append the file PART2 found on the default unit to file PART1 found on unit two.

COPY ALLPARTS=PART1,D3:PART2,PART3

Transfer the contents of PART1 to ALLPARTS, then append PART2 on unit three to the new file ALLPARTS, and finally, append PART3 to ALLPARTS.

NOTES:

In the event that an error occurs, the output file may be left open and should be closed using the CLOSE command.

DIR

Display list of files on disk.

STATUS:

User memory at PAD is untouched.
This command is interruptable.

COMMAND FORMATS:

DIR {filespec}
DIR unit

OPERATION:

The DIR command lists all files within the directory on the default drive unit unless an optional file specification or drive number is specified. If the optional filespec is specified, it must resolve to a legal filename or else an error will result. Likewise, if the optional unit specification is supplied, it must be a number from 1 to 4 inclusive. The DIR command also displays the current number of free sectors. Note that the size of a file displayed in the listing is not necessarily accurate unless that file is closed. However, the number of free sectors on disk is always accurate.

EXAMPLES:

DIR	list information about all files found on the default unit.
DIR MYFILE	list information on MYFILE found on the default unit.
DIR D2:*.4TH	list information on all files with the extension 4TH found on unit two.
DIR 3	list information about all files found on unit three.

NOTES:

If the DIR command is given within a file that is to be loaded, a filespec or unit must be specified. Thus the first example above must be: DIR *

DISKCOPYn

Duplicate an existing diskette.

STATUS:

PAD is modified.

COMMAND FORMAT:

DISKCOPY1	single-drive copy
DISKCOPY2	multi-drive copy

OPERATION:

DISKCOPY1 is for users with only one disk drive. DISKCOPY2 is for users with two or more drives.

EXAMPLES:

DISKCOPY1

Copy a disk using only drive one. The user is prompted to insert the source diskette, and then the destination diskette. This is repeated until the entire source diskette is duplicated.

DISKCOPY2

The diskette in unit one is copied to the diskette in unit two.

NOTES:

EDIT

Edit a (FORTH) source file.

STATUS:

PAD is modified.

COMMAND FORMAT:

EDIT infile(,outfile)

OPERATION:

The input file is read into memory beginning at PAD where the editor is used to modify it. Upon leaving the editor, the modified file is written to the output file, if specified, otherwise it is written back to the input file. See the valDOS File Editor documentation in section LXIV for further information.

EXAMPLES:

EDIT MYFILE

Edit the file MYFILE on the default drive unit and write the modified version back to MYFILE.

EDIT D1:PACMAN.4TH,D2:PACMAN.4TH

Edit the file PACMAN.4TH on unit one and write the resultant file into file PACMAN.4TH on unit two.

NOTES:

ENDFIL

Endfile at current file cursor position.

STATUS:

User memory at PAD is untouched.

COMMAND FORMAT:

ENDFIL filenum

OPERATION:

The current file cursor position within the specified file is marked as the new end of that file. All data after that point in the file is lost and any disk space used by lost data is reclaimed.

EXAMPLE:

ENDFIL 2

End-file file number two.

NOTES:

Care should be taken if this command is used on a file that is open under more than one file number.

EOF

Move the file cursor to the end of the file.

STATUS:

User memory at PAD is untouched.

COMMAND FORMAT:

EOF filenum

OPERATION:

The cursor of the specified file is repositioned at the end of the file. The file must already be open.

EXAMPLE:

EOF 1

Position the cursor of file one at the end of that file.

NOTES:

ENTER

Enter (create) a filename in a disk directory.

STATUS:

User memory at PAD is untouched.

COMMAND FORMAT:

ENTER filename

OPERATION:

The specified filename is entered into the indicated directory. The filename must not already exist or an error will result. The file is created, but is not opened. Although this command is usually called CREATE, that word already exists in FORTH.

EXAMPLES:

ENTER MYFILE

Enter the filename MYFILE into the directory on the default drive unit.

ENTER D2:INVADERS.4TH

Enter the filename INVADERS.4TH into the directory on unit two.

NOTES:

FDUMP

Perform a hex/ASCII dump of a file.

STATUS:

User memory at PAD is untouched.
This command is interruptable.

COMMAND FORMAT:

FDUMP filename(/W)

OPERATION:

The specified file is displayed as a sequence of hex numbers and ASCII equivalents. This is typically used for looking at machine language programs stored on disk. Normal output is 8 bytes per line, however, if the /W (for "wide") switch is present, 16 bytes per line are displayed, which is more suitable for printed output.

EXAMPLES:

FDUMP MYFILE.OBJ

Dump the file MYFILE.OBJ to the current output device. Eight bytes/line are displayed.

FDUMP MYFILE.OBJ/W

Dump the file MYFILE.OBJ to the current output device. 16 bytes/line are displayed.

NOTES:

FILE-IT

Transform FORTH screen format to DOS file format

STATUS:

PAD is used.

COMMAND FORMAT:

```
FILE-IT  scr1,scr2,filename
```

OPERATION:

The screens from scr1 to scr2, inclusive, are read into free memory. The DOS disk is then swapped into the drive and the screens are written to the specified filename.

EXAMPLES:

```
FILE-IT  50,60,MYCODE.4TH
```

Screens 50 through 60 are read into free memory. The user is then prompted to insert the DOS-format disk into the drive. The data is next written to the file MYCODE.4TH.

NOTES:

FLOAD

Compile a FORTH source file from disk.

STATUS:

PAD is moved by compilation as usual.
This command is interruptable.

COMMAND FORMAT:

FLOAD filename{/C}
FLOAD filename{,linenum}

OPERATION:

The FLOAD command sends the FORTH source code contained in the specified file to the valFORTH compiler. If the /C (for "continue") switch is given, loading begins at the beginning of the last line edited in the file editor. This allows load errors to be fixed and compilation to be continued mid-file. If the optional line number is present, loading proceeds from that line of the file.

EXAMPLES:

FLOAD MYGAME.4TH

Load the DOS file MYGAME.4TH from the default drive unit and compile it.

FLOAD D2:CYCLOPS/C

Load the DOS file CYCLOPS from unit two starting with the last line edited in the file editor.

FLOAD FORTRAN.4TH,50

Load the DOS file FORTRAN.4TH from the default unit beginning with the 50th line in the file.

NOTES:

It is possible to LOAD a screen from a file being FLOADed and vice versa. Usually, LOADing and FLOADing should done from different units. Also see ECHO command.

FMOVE

Single drive interdisk file transfer

STATUS:

Memory at PAD is used.

COMMAND FORMAT:

FMOVE outfile=infile

OPERATION:

The FMOVE command is for those users who have access to only one disk drive. It is used to transfer a DOS file from one disk to another. If infile is too large to fit in available free memory, multiple disk swaps must be made. FMOVE will prompt for all necessary inputs, and swaps.

EXAMPLES:

FMOVE MYFILE.BAK=MYFILE

Transfer contents of MYFILE to MYFILE.BAK. MYFILE is read into free memory, source and destination disks are then swapped, and the data stored in free memory is written to MYFILE.BAK.

NOTES:

Multi-drive users can use the COPY command for this purpose.

FORMAT

Format a diskette for use.

STATUS:

User memory between PAD and BUFBOT is untouched.

COMMAND FORMAT:

FORMAT <unit>

OPERATION:

The FORMAT command initializes a disk for use with valDOS. If a unit number is supplied, formatting will be attempted on that unit, otherwise the default unit is assumed. The command will issue verification prompts and will also allow sectors to be locked so that no file will ever occupy those sectors. This feature allows mixing FORTH screens (virtual memory) and files on a single disk. The valDOS II disk is an example of this. This command also allows the newly formatted disk to be named.

EXAMPLES:

FORMAT

Format the default drive unit.

FORMAT 2

Format unit two.

NOTES:

This command replaces the one found on the valFORTH 1.1 disk. See NAMEDISK.

FSPACE

Move the file cursor relative to its current position.

STATUS:

User memory between PAD and BUFBOT is untouched.

COMMAND FORMAT:

FSPACE filenum,count

OPERATION:

The cursor of the specified file is moved by the signed offset specified by count. The file cursor may be moved up to 32767 bytes in either direction, with a negative count spacing toward the beginning of the file. If the count is zero, the file cursor is left untouched.

EXAMPLES:

FSPACE 1,123

Move the cursor of file one 123 bytes toward the end of the file.

FSPACE 2,-345

Move the cursor of file two 345 bytes toward the start of the file.

NOTES:

Spacing backward is generally much slower than spacing forward due to the manner in which data is stored on disk. For this reason, backward spacing should be avoided.

KILL

Remove files and release disk space.

STATUS:

User memory between PAD and BUFBOT is untouched.

COMMAND FORMAT:

KILL filespec1(/N)(,filespec2(/N)(,...))

OPERATION:

The KILL command removes the specified files from the specified unit. If no unit is given in the file specification, the default unit is assumed. If the filespec is unambiguous (i.e., no wild cards), no verify prompt is issued. If the filespec is ambiguous, a verify prompt is issued for every file about to be deleted unless the /N switch is present.

EXAMPLES:

KILL THIS

Delete file THIS on the default drive unit.

KILL PART1,PART2,PART3,D3:PART?

Remove files PART1, PART2, and PART3 from the default unit, and all files found on unit three with five letter names that begin with PART.

KILL *.BAS,*.BAK/N,D2:MYFILE,TEST.*/N

Remove all files on the default unit with the extension BAS giving verify prompts. Remove all files with the extension BAK without verify prompts. Delete MYFILE on unit two, and all files named TEST on the default unit.

NOTES:

LOCK

Write and modify-protect files on disk

STATUS:

User memory at PAD is untouched.

COMMAND FORMAT:

LOCK filespec1{/N}{,filespec2{/N}{,...}}

OPERATION:

The LOCK command protects the specified files from being modified in any way. If the filespec is unambiguous (i.e., no wild cards), no verify prompt is issued. If the filespec is ambiguous, a verify prompt is issued for every file about to be locked unless the /N switch is present.

EXAMPLES:

LOCK D2:*

Lock all files on unit two, with prompts.

LOCK FILE?/N,PART1,PART2

Lock all files with five letter names that start with FILE, without prompts. Lock PART1 and PART2.

LOCK *.4TH,D2:MYFILE,PROG*/N

Lock all files with the extension 4TH giving verify prompts. Lock MYFILE on unit two, and all files beginning with PROG on the default unit, without prompts.

NOTES:

NAMEDISK

Name a diskette.

STATUS:

User memory at PAD is untouched.

COMMAND FORMAT:

NAMEDISK <unit>

OPERATION:

The valDOS file system allows disks to named for identification. Currently, this name is displayed only in directory listings, but is available for user programs. The NAMEDISK command displays the name of the disk in the specified unit and prompts for the new name to be entered. Disknames may be up to 20 characters long, and any character may be included within that name.

EXAMPLE:

NAMEDISK 3

Rename the disk in unit three.

NOTES:

Disks named in valDOS will function properly in other DOS's for the Atari computers.

OPEN

Open a file for access.

STATUS:

User memory at PAD is untouched.

COMMAND FORMAT:

OPEN filename

OPERATION:

The OPEN command assigns a buffer area and a file access number to the specified file.

EXAMPLE:

OPEN TEST.4TH

Opens the file TEST.4TH on the default unit.

NOTES:

Files may be multiply open, but are logically different files as far as the DOS is concerned. If a file is opened more than once and an operation such as ENDFIL is given, it is possible that the other opens will contain data in their transfer buffers that technically no longer exists. Note also that disks should not be exchanged when there are files open on the disk.

OPEN?

List all files currently open.

STATUS:

User memory at PAD is untouched.

OPERATION:

All files currently open are displayed along with their associated file access numbers.

EXAMPLE:

OPEN?

NOTES:

PRINT

Display a text file on the current output device.

STATUS:

User memory at PAD is untouched.
This command is interruptable.

COMMAND FORMAT:

PRINT filename{/N}{,linenum}

OPERATION:

The contents of the specified file are sent to the current output device. Each line of text is automatically numbered unless the /N switch is present. If the optional line number specification is supplied, printing will begin with that line of the file.

EXAMPLES:

PRINT GALAXY.4TH

Print the file GALAXY.4TH with line numbers.

PRINT EDITOR.4TH/N

Print the file EDITOR.4TH without line numbers.

PRINT D2:MYFILE.TXT,56

Print the file MYFILE.TXT on unit two with line numbers, beginning with the 56th line of text in the file.

NOTES:

READ

Read a file into memory.

STATUS:

User memory between PAD and BUFBOT is untouched.

COMMAND FORMATS:

```
READ  filename,address(,count)
READ  filenum,address(,count)
```

OPERATION:

In the first form, the specified file is opened and the first "count" bytes are read into memory starting at the address specified. If the count is not specified, the entire file is read in. The file is left closed. In the second form of the command, the first "count" bytes of the already opened file are read into the specified address. The file is left open after the read is complete. Note that the address can be specified by a number or by a single word (such as PAD) which returns a number.

EXAMPLES:

```
READ  DRIVER.OBJ,PAD
```

Read the entire file DRIVER.OBJ into the address specified by PAD. The file is then closed.

```
READ  1,40960,1000
```

Read the first 1000 bytes from the file specified by the file access number one into address 40960. The file is left open.

NOTES:

No check is made to see if the data is being read into memory occupied by the FORTH dictionary, DOS buffers, or video memory. It is up to the user to supply safe load addresses.

RENAME

Rename a file.

STATUS:

User memory at PAD is untouched.

COMMAND FORTMAT:

RENAME newname=oldname

OPERATION:

The specified file is given the specified new name. There must not already exist a file with the same name as the specified new name or an error will result.

EXAMPLE:

RENAME MYFILE.4TH=MYFILE

The file MYFILE is renamed as MYFILE.4TH.

NOTES:

REWIND

Move the file cursor to the beginning of the file.

STATUS:

User memory at PAD is untouched.

COMMAND FORMAT:

REWIND filenum

OPERATION:

The cursor of the specified file is repositioned at the beginning of the file. The file must already be open.

EXAMPLE:

REWIND 1

Rewind file number one.

NOTES:

SETUNIT

Set the default drive unit.

STATUS:

User memory at PAD is untouched.

COMMAND FORMAT:

SETUNIT unit

OPERATION:

Whenever a filename does not explicitly contain a drive specification, the new default drive unit will be assumed. Units are numbered from one to four.

EXAMPLE:

SETUNIT 2

Set the default unit to two.

NOTES:

UNLOCK

Unprotect a file so that it may be modified.

STATUS:

User memory at PAD is untouched.

COMMAND FORMAT:

UNLOCK filespec1(/N)(,filespec2(/N)(,...))

OPERATION:

The UNLOCK command removes protection from the specified files so that they may be killed or modified. If the filespec is unambiguous (i.e., no wild cards), no verify prompt is issued. If the filespec is ambiguous, a verify prompt is issued for every file about to be unlocked unless the /N switch is present.

EXAMPLES:

UNLOCK THIS

Unlock file THIS on the default drive unit.

UNLOCK PART1,PART2,PART3,D3:PART?

Unlock files PART1, PART2, and PART3 from the default unit, and all files found on unit three with five letter names that begin with PART.

UNLOCK *.BAS,*.BAK/N,D2:MYFILE,TEST.*/N

Unlock all files with the extension BAS giving verify prompts. Unlock all files with the extension BAK without verify prompts. Unlock MYFILE on unit two, and all files named TEST on the default unit.

NOTES:

WRITE

Write an area of memory to a file.

STATUS:

User memory at PAD is untouched.

COMMAND FORMATS:

```
WRITE  filename,address,count
WRITE  filenum,address,count
```

OPERATION:

In the first form, the first "count" bytes of memory at the specified address are written to the specified file. If the file does not already exist, it is created. The file is closed after the write. In the second form, the memory block is written to the already open file. The file is left open. The address can be specified by either a number or a single word (such as PAD) which returns a number.

EXAMPLES:

```
WRITE  MYFILE,PAD,1000
```

Write the 1000 byte block of memory at the address specified by PAD to MYFILE.

```
WRITE  1,40960,256
```

Write the first 256 bytes of memory at address 40960 to the file associated with file access number one.

NOTES:

The valDOS System

The following set of words make up the heart of the valDOS system. Note that all of the following words are in a special vocabulary named DOS. Any word which uses one of these system operations must contain the word DOS in the definition, as it is necessary to inform the compiler where to look for these definitions. Since most of these commands are on the system level, a brief description of how the system works is appropriate.

Before any file can be manipulated, it must be opened for access. A file buffer and a status block are allocated for each open file. Additionally, each file has its own file cursor (which is positioned at the beginning of a file at open time). This cursor always points to a location within the file. If a byte is read from the file, the byte pointed to by this cursor is returned and the cursor is moved a byte deeper into the file. When a byte is written to the file, it replaces the byte pointed to by the file cursor, and the file cursor is then bumped.

Thus, if a 10K file is opened and five bytes are written to the file, the first five bytes will be changed, and the remainder of the file will be left untouched. If the file is then closed, the file will still be 10K long. Most BASICs, however, will "end-file" a file when it is closed, i.e., in this example, the file would be reduced to five bytes. By not implicitly end-filing a file when it is closed, greater flexibility is gained. Note that the valDOS commands (such as COPY and EDIT) do implicitly perform an end-file prior to closing a file.

The two most common operations are reading a file and writing a file. Usually when reading a file, the file is opened, read to the end (eof), and then closed. Generally when a file is written, it is opened, written, end-filed, and then closed.

valDOS System Words

DOS ---

This is the name of the vocabulary which contains all of the following DOS system words. DOS stands for Disk Operating System (not Software as some claim). Any word which contains one of the following words must contain this word within its definition: : myword DOS ... ; DOS is IMMEDIATE.

#ENTRIES --- n

A quan which contains the number of entries that matched the (wild) filename last checked by CHKDIR. CHKDIR returns this value automatically. This value can also be used to index the first "n" elements of the table pointed to by DIRTBL (below).

#FREE --- n

A quan which contains the number of free directory entries in the last directory scanned by CHKDIR.

#UNTRN --- n

A quan which contains the number of bytes left untransferred in the last block read/write operation. This value is only accurate immediately after the read/write operation returns control to the calling routine.

(?OPEN) $ --- (f 1)/0

This routine checks to see if the file specified by $ is open. If the operation is successful, a flag is returned along with the value one. If this flag is 0, the file is closed. If an error occurred, only a zero is returned. See DSKERR. This routine uses FNCON.

(OPEN) $ --- (addr fl# 1)/0

This routine opens the file specified by $. It is used to create the necessary data transfer path between the application and the DOS. If the open is successful, a file transfer buffer is allocated and is assigned a file access number. All subsequent operations upon the open file require that this file number be supplied. On a successful open, the file number and the transfer buffer address are returned. In most cases, the buffer address can simply be discarded, while the file number must be stored by the application. File numbers are always greater than zero. See DSKERR. This routine uses FNCON.

(CLOSE) fl# ---

The (CLOSE) operation closes the data transfer path associated with the specified file number. The file buffer is flushed to disk if updated, and is freed for a subsequent open operation. If "fl#" is zero, all open channels are closed from access. This routine generates no errors. Illegal file numbers are ignored.

(ENDF) fl# ---

Each open file has a file cursor which points to the next byte to be read or written. The "end file" operation marks the current byte as the end of the file. Thus if the file cursor points to the fifth byte of a 10K file, the current byte and all successive bytes are lost, and the disk space is reclaimed. This is typically used just before closing a file that has been written to. This ensures that no "stale" data remains. The command ENDFIL uses this routine.

(ENTER) $ --- f

The filename specified by $ is entered into the directory on the unit specified within the filename. If no unit is explicitly stated, the default unit (specified by DFLUNT) is assumed. A one is returned if the operation was successful, otherwise a zero is returned. See DSKERR. This routine uses FNCON. (This routine is usually called "create". That name is already used in FORTH, however.)

(KILL) $ -- f

The filename specified by $ is deleted from the directory on the unit specified within the filename. If no unit is explicitly stated, the default unit (specified by DFLUNT) is assumed. A one is returned if the operation was successful, otherwise a zero is returned. See DSKERR. This routine uses FNCON.

(LOCK) $ --- f

The filename specified by $ is locked so that it may not be written to, killed, or renamed. A one is returned if the operation was successful, otherwise a zero is returned. See DSKERR. This routine uses FNCON.

(RDB) fl# --- (b 1)/O

The "read byte" operation reads the next byte from the file whose access number is "fl#". If an error occurs a zero is returned, otherwise the byte along with a one is returned. If many bytes are to be read, (READ) should be used if possible as that routine is many times faster than (RDB). Note that if a read is attempted with the file cursor at the end of the file, an EOFERR error is generated. See DSKERR.

(READ) addr cnt fl# --- f

This operation reads the next "cnt" bytes of the file whose access number is "fl#" and stores them in memory beginning at "addr". If an error occurs a zero is returned, otherwise a one is returned. The quan #UNTRN contains the number of bytes left untransferred in the event of an error. See DSKERR.

(REN) $n $o --- f

This operation renames the file $o to $n. A one is returned if no error occurred, otherwise a zero is returned.

(SPACE) cnt fl# --- f

The space operation repositions the cursor of the file whose access number is "fl#" by the signed number "cnt". The value "cnt" must lie in the range of -32768 to 32767. Also note that the file cursor cannot be spaced past byte no. 65,535 of the file. If "cnt" is zero, the space operation is ignored. A one is returned if no error occurred, otherwise a zero is returned.

(UNLOCK) $ --- f

The filename specified by $ is unlocked so that it may be written to, killed, or renamed. A one is returned if the operation was successful, otherwise a zero is returned. See DSKERR. This routine uses FNCON.

(WIND) f fl# --- f

The (WIND) command is used to position the file cursor at beginning or end of the file. If "f" is one, the file cursor is rewound to the beginning of the file. This allows the file to be re-read. If "f" is zero, the file cursor is moved to the end of the file for subsequent writing, effecting an append operation. A one is returned if the operation was successful, otherwise a zero is returned. See DSKERR.

(WRB) b fl# --- f

The byte "b" is written to the file whose access number is "fl#". If many bytes are to be written to the file, (WRITE) below should be used instead, as it is many times faster than (WRB). A one is returned if the operation was successful, otherwise a zero is returned. See DSKERR.

(WRITE) addr cnt fl# --- f

The block of memory "cnt" bytes long beginning at memory location "addr" is written to the file whose access number is "fl#". A one is returned if the operation was successful, otherwise a zero is returned. See DSKERR.

?DOSERR f err# ---

This is one of three (see DOSERR, ?DSKERR) error routines available at the system level. If the flag "f" is zero, the system error "err#" is generated. Program control does not return to the word which contained ?DOSERR, but to the word which called this word. For example, if an application calls (OPEN) and a ?DOSERR within (OPEN) generates an error, ?DOSERR will not return to (OPEN), but to the application, passing a 0. If the ?DOSERR does not generate an error, program control will return to (OPEN). ?DOSERR stores the error value in the quan DSKERR. See the actual definition of (OPEN) for a good example of how this is used. ?DOSERR is essentially a SWAP 0= IF DOSERR ENDIF DROP

?DSKERR f ---

?DSKERR is used to propogate an error from one word to the next. It replaces the sequence DSKERR ?DOSERR. See ?DOSERR.

?WILD --- f

This is a quan which contains a one if the last filename converted by FNCON contains either of the wild card characters "?" or "*". A zero is returned if no wild cards appeared.

BUFBOT --- addr

This is a word which returns the lowest memory address used by the DOS file buffers.

CHKDIR --- n

The check-directory routine scans the directory on the unit specified within the last filename converted by FNCON for all occurrences of that filename. For every match found (multiple matches are due to wild cards) in the directory, a directory entry number is stored in the next available location of the memory block pointed to by DIRTBL. Thus if CHKDIR finds five occurrences of a filespec, the first five elements of DIRTBL contain the directory entry numbers for those five matches. These values can then be used in conjunction with the ENTRY command to access the files. CHKDIR returns the number of matches.

DFLUNT --- n

A quan which contains the number of the default unit. This drive number is assumed if no drive specification is contained within a filename converted by FNCON. Caution, this contains 0 if DOS drive 1 is the default, 1 if DOS drive 2 is the default, etc. (FORTH drive = DOS drive-1)

DIRFRE --- n

A quan which contains the entry number of the next free entry in the directory last scanned by the CHKDIR routine.

DIRTBL --- addr

A word which returns the starting address of a 64 byte memory block that contains directory entry numbers of all files that matched the last filename checked by the CHKDIR routine. See CHKDIR.

DIRUP ---

This marks the current directory as being updated so that it is written to disk upon the next DSKFLS command.

DOSERR err# --- 0

This is one of three (see ?DOSERR, ?DSKERR) error routines at the system level. DOSERR unconditionally generates the error whose error number is "err#". Program control continues two levels up (instead of the usual one) and a zero is returned. See ?DOSERR.

DSKERR --- n

A quan which contains the error number of the last DOS error that occurred. See List of Errors below.

DSKFLS ---

A routine which flushes the current directory and free space map if updated. All user defined commands should end with this command. See source listing for pre-defined commands.

ENTRY unit n --- addr

The entry command returns the address of the "nth" entry in the directory on the specified drive unit. The 16 byte entry has the following format:

```
addr+0:   status byte
          bit 7:  File deleted if set
              6:  File entry valid if set
              5:  File locked if set
              4:  File random if set
              3:  File in use if set (not used)
              2:  Unused
              1:  DOS format 2 if set
              0:  File open for output if set
addr+1:   length of file in sectors
addr+3:   first (DOS) sector of file
addr+5:   8 letter filename,
          left justified, blank filled
addr+13:  3 letter extension,
          left justified, blank filled
```

If any changes are made, executing DIRUP and DSKFLS will write those changes to disk.

FLBUF@ fl# --- addr

Whenever a file is opened, a 128 byte transfer buffer and a 16 byte status block are allocated. The FLBUF@ command returns the address of the file status block associated with "fl#". The 16 byte table contains the following information:

```
addr+0:    File status byte (see ENTRY).
addr+1:    Current size of file in sectors.
           If high bit is set, file is updated.
addr+3:    First (DOS) sector of file.
addr+5:    (FORTH) sector currently in file buffer.
addr+7:    Number of bytes into current sector.
addr+8:    Unit associated with the file.
addr+9:    Entry number in the directory.
addr+10:   Non-zero = current sector is updated.
addr+11:   (reserved for) Current random block.
addr+13:   (reserved for) Random block update flag.
addr+14:   Number of bytes into file (unsigned).
addr-128:  Address of 128 byte file buffer.
```

FLFLS fl# ---

This operation flushes the file buffer associated with the file access number "fl#".

FNCON $ --- f

The FNCON command takes the file specification $ and converts it to directory format (i.e., left justified and blank filled). It stores the 11 byte formatted filename at the memory location pointed to by FNFLD. All wild cards are converted to question marks, thus, "MY?FIL*" will become "MY?FIL?????". Additionally, if a drive specification is contained in the filespec, it is determined and stored in the quan UNIT. If no drive specification was supplied, the default unit (in DFLUNT) is stored in UNIT. FNCON will also parse out a single switch ("/s") where the character "s" is stored in the quan FNSWCH. If no switch is found, a zero is stored in FNSWCH. If any wild cards appear in the file specification, the quan ?WILD is set to one, otherwise zero is stored. If no errors were detected, a one is returned, otherwise zero.

FNFLD --- addr

A pointer to an eleven byte storage area which contains the filename last formatted by FNCON.

FNSWCH --- s

A quan which contains the ASCII value of the switch in the last filename converted by FNCON. If no switch was present, FNSWCH contains zero. See FNCON.

FSMAP --- addr

A word which returns the address of the current free space map.

FSMUP ---

The FSMUP command marks the current free space map as being updated.

MAXFL ---

A constant which contains the maximum number of files that can be open at any given time. This has a default value of four. If this constant is changed, valDOS must be completely reloaded.

TADR --- n

After a block read/write operation, this quan contains an address one byte higher in memory than that of the last byte transferred. This can be used to determine how much data was transferred in the event of an error.

UNIT --- n

A quan which contains the unit specification of the last filename converted by FNCON. Note that this is a FORTH unit (i.e., FORTH unit = DOS unit-1).

WRKSPC --- addr

A quan which points to a 128 byte scratch area used by many of the system words described above. This is free for user applications between valDOS system calls.

valDOS Command Support Words

?CMDERR n err# ---

Like ?DOSERR, but prints the error message, clears both stacks, and aborts program execution through QUIT, if the status flag "n" is zero. If "n" is true, ?CMDERR returns to the calling word.

?SYSERR n ---

Like ?DSKERR, but aborts through ?CMDERR if "n" is zero. If "n" is non-zero, ?SYSERR returns to the calling word.

?WRGARG n ---

This is an abbreviation for WRGARG ?CMDERR .

CONFN addr unit --- $

The CONFN routine takes the directory formatted filename at "addr" (usually FNFLD) and converts it to a string. The drive specification "unit" is attached to the beginning of the filename (i.e., unit = 0 would generate "D1:").

CMDERR err# ---

Like DOSERR, CMDERR unconditionally generates the error specified by "err#".

ECHO ON/OFF ---

When echo is ON, all files being FLOADed will be echoed to the current output device(s). When echo is OFF, no output is generated.

FWORD c ---

Read the next text characters from the input file whose file number is in the quan FLFL# (FLOAD initializes FLFL#) until a delimiter "c" is found, storing the packed character string beginning at the dictionary buffer HERE. FWORD leaves the character count in the first byte, followed by the characters, and ends with two or more blanks. Leading occurrences of "c" are ignored. Note that "c" may not be the return character (ATASCII 155).

GETVAL $ --- n

If $ is the name of a dictionary word, the word is executed and had better return a single value. If $ is not found in the dictionary, it is assumed to be a number and is converted leaving the number n. GETVAL is used so that addresses may be stated explicitly, or by reference (PAD, HERE, etc.).

GETARGS --- {$ 1}/0

This routine converts the next non-blank set of characters in the input stream to a string. If a set of non-blank characters is found, it is returned along with a one. If no set is found, a zero is returned.

GETARG $ c --- {$1 $2 1}/{$ 0}

The string $ is divided into two parts, broken at the first occurrence of the character "c". If the character "c" is found in $, the leftmost portion of $ is returned 3rd on stack, the rightmost 2nd on stack, and 1 on top. If "c" is not in the string, the original string along with 0 is returned.

System Errors (all are CONSTANTs)

AMBNME --- n

Ambiguous filename. This error is generally issued when a filespec containing wild cards is passed to one of the primitive file operators like (OPEN) or (KILL).

BADFL# --- n

Bad file number. The error is generated when a file number does not lie in the range 1 ≤ fl# ≤ MAXFL (default of four).

BADFSM --- n

Bad free space map. This is issued when the number of free sectors does not match the true number of free sectors. If this error is reported, a new disk should be made and all files should be transfered to this new disk using COPY or FMOVE.

BADNME --- n

Bad filename. The filename passed to the system routine contained an illegal character, or was too long. File names can only contain the letters "A" through "Z" and the digits "0" through "9". Note that the first character of a filename must be a letter. The two wild card characters "?" and "*" are also allowed.

BADUNT --- n

Bad drive specification. A drive number was encountered that did not lie in the range 1 ≤ unit ≤ 4.

DIRFUL --- n

Directory is full. There is no more room in the directory. Kill some unwanted files and try the operation again.

DSKFUL --- n

Disk is full. There are no more free sectors on the disk. Kill some unwanted files and try the operation again.

EOFERR --- n

End of file has been reached. This error generally results from an attempt to read data past the end of file mark.

FLDNE --- n

File does not exist.

FLEXST --- n

File already exists.

FLNOPN --- n

File is not open. Use the OPEN command and open the file. This error should only occur when using the file primitives like (SPACE) and (WIND).

FLOPN --- n

File is open. Use the CLOSE command to close the appropriate file. See OPEN?

FLTBG --- n

File is too big. This error is usually reported by the file editor when there is not enough free memory to edit the file. Break the file into two parts, or use the FORGET command to free some memory.

FLWPRT --- n

file is write protected (locked). Use the UNLOCK command and perform the last operation again.

TMFOPN --- n

Too many files open. For each file open, a file buffer and a file status block is allocated. There are a limited number of available buffers (determined by the constant MAXFL). If all buffers are being used and an open operation is attempted, this error will be generated. Use the CLOSE command to free a buffer.

WRGARG --- n

Bad/no argument list. This is generated when a DOS command expects a list of arguments and none is supplied. This is also generated when the wrong number or type of arguments is supplied.

A note on QUAN structures

The "quan" is a new FORTH data structure, developed at Valpar, and is used in this package. Quans were devised to cut down on wasted memory and runtime encountered when using the "variable" data structure. Quans work as follows: (Advanced users may want to follow along in the source code for these structures also.

Defining a quan:
QUAN BINGO

Note that quans do not take initial values. This form was chosen to allow for simpler upgrading to target-compiled code later on.

Giving a quan a value:
1234 TO BINGO

Note that since TO is immediate, "TO BINGO" compiles to only 2 bytes instead of the 4 bytes that would be required if BINGO were a variable (i.e., BINGO !).

Getting the value back from a quan:
BINGO

Simply saying the name of the quan will leave its value on the stack, in this case 1234. In this way, quans act like constants. BINGO above also compiles to only 2 bytes instead of the 4 bytes required to fetch it if it were a variable (i.e., BINGO @).

Getting the address of the data in the quan:
AT BINGO

This will leave the address of the first byte of data in BINGO on the stack, or compile the address as a literal if encountered during compilation. (AT is immediate.) This is useful for a variety of purposes in general programming and in interfacing to machine language routines.

Advanced users:

The FORTH 83 Standard appears to lean toward "non-state-smart" words, which is proper for target-compiled applications. We expect to support both "state-smart" and "non-state-smart" versions of various words, as appropriate for different users.

Note that while

15 AT BINGO +! and 15 BINGO + TO BINGO

accomplish the same task and take the same amount of memory, the first version is faster by one primitive nest.

The most significant internal feature of a quan is that it has 3 cfa's instead of just the one common to most FORTH words.

This initial four byte disadvantage is overcome at the second use of a quan, and so poses essentially no problem. When a quan is not preceded by "TO" or "AT", the first cfa (quan@) is compiled in. If the quan is preceded by "TO", the second cfa (quan!) is compiled in. And if the quan is preceded by "AT", the third cfa ('quan) is compiled into the dictionary.

valDOS File Editor

Version 1.0
Oct. 1982

The FORTH language is a very powerful addition to the Atari home computer. Programs which are impossible to write in BASIC (usually because of limitations in speed and flexibility) can almost always be written in FORTH. Even when one has mastered the BASIC language, making corrections or additions to programs can be tedious. The video editor described here removes this problem from the FORTH environment. Similar to the MEMO PAD function in the Atari operating system, this editor makes it possible to insert and delete entire lines of code, insert and delete single characters, toggle between insert and replace modes, move entire blocks of text, and much more.

Overview

This editor is a powerful extension to the valFORTH system designed specifically for the Atari 400/800 series of microcomputers. The main purpose for this editor is to give the FORTH programmer an easy method of text entry to DOS file for subsequent compilation. For those already familiar with the valFORTH 1.1 screen editor, this editor is very similar in function. In fact, all of the commands found in that editor (except ctrl-A) are supported in the file editor. More importantly, many additional capabilities have been added to this editor. They are:

1) Tab stops can be set/reset
2) Splice (unsplit) is now supported
3) Global pattern searches
4) A repeat function which repeats the next command/key typed until a console key is pressed
5) File merge (i.e., reading a file into a file)
6) True single-/mult- line scrolling either forward or backward
7) Input and output files may be different

The editor has four basic modes of operation:

1) It allows entering of new text into a file as though typing on a regular typwriter.

2) It allows simple modification of any text with a powerful set of single stroke editing commands.

3) It pinpoints exactly where a compilation error has occurred and sets up the editor for immediate correction and recompilation.

The set of single stroke editing commands is a superset of the functions found the in MEMO PAD function of the standard Atari operating system. In addition to cursor movement, single character insertion/deletion, and line insertion/deletion, the editor supports a clear-to-end-of-line function, a split command which separates a single line into two lines, its complement splice (unsplit), global searches, and many other features usually found only in higher quality word processors, and almost never in file editors.

Also provided is a visible edit storage buffer which allows the user to move, replace, and insert up to 320 lines of text at a time. This feature alone allows the FORTH programmer to easily reorganize source code with the added benefit of

knowing that re-typing mistakes are avoided. Usage has shown that once edit-buffer management is learned, significant typing and programming time can be saved.

For those times when not programming, the editor can double as a simple word processor for writing letters and filling other documentation needs. Perhaps the best method for learning how to use this powerful editor is to enter the edit mode and try each of the following commands as they are encountered in the reading.

NOTE:

This editor can be used to enter assembly language source, Pascal, or any other text oriented data. The only limitation upon this is that no lines may be longer than 38 characters in length. Additionally, this editor can edit files created from other sources; however, only the first 38 characters of a line will be retained.

Loading and Entering the Editor

To load the editor, first load valDOS as described in "Strolling Through valDOS". Next, insert the valDOS II disk and list screen 170. This should tell you exactly which screen to load. The edit mode is initiated using the EDIT command. This command has the following format:

```
EDIT  infile(,outfile)
```

The DOS file "infile" is loaded for editing. When all changes have been made and the file is saved, the modified text is written to the file "outfile", if supplied, otherwise it is written back to "infile".

Insert a copy of your valDOS II disk and type:

```
EDIT  FILEIT.4TH,MYFILE
```

The editor will display some information which can be ignored for the time being, and then it will wait for the return key to be typed. After pressing the return key, the display should look like fig. 1.

The top window, composed of a single line, indicates which file is currently being edited. Also shown is the size of the edit buffer (decribed later). In this example, the buffer is five lines in length. This window is known as the heading window.

The second window (the text window) contains 16 lines of text within the specified file. This window is 38 characters wide and 16 lines high. The white cursor (indicated by the symbol "▒") will be in the upper-lefthand corner of the display awaiting editing commands.

The final five-line window found at the bottom of the display is known as the buffer window. This is used for advanced editing and is described in greater detail in the section entitled "Buffer Management."

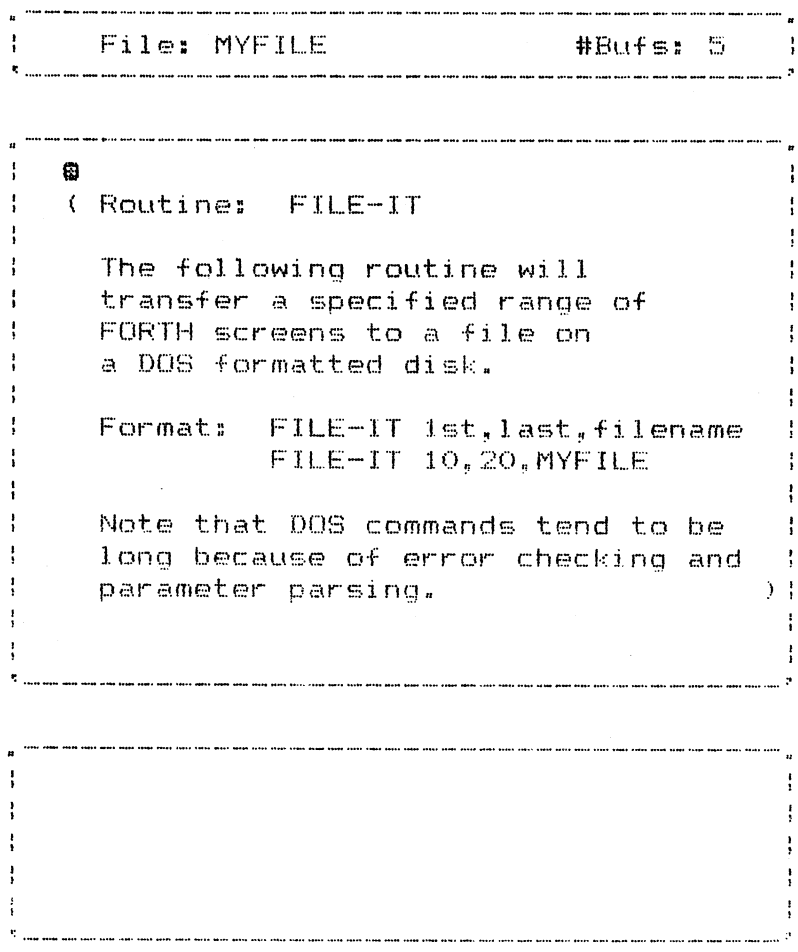


Fig. 1

LL re-edit last file (---)

This command is used to re-edit the "Last" file edited. It functions identically to the EDIT command, except that no file names need to be specified.

Example: LL <ret> (re-edit MYFILE)

Loading and Entering the Editor

To load the editor, first load valDOS as described in "Strolling Through valDOS". Next, insert the valDOS II disk and list screen 170. This should tell you exactly which screen to load. The edit mode is initiated using the EDIT command. This command has the following format:

```
EDIT  infile{,outfile}
```

The DOS file "infile" is loaded for editing. When all changes have been made and the file is saved, the modified text is written to the file "outfile", if supplied, otherwise it is written back to "infile".

Insert a copy of your valDOS II disk and type:

```
EDIT  FILEIT.4TH,MYFILE
```

The editor will display some information which can be ignored for the time being, and then it will wait for the return key to be typed. After pressing the return key, the display should look like fig. 1.

The top window, composed of a single line, indicates which file is currently being edited. Also shown is the size of the edit buffer (decribed later). In this example, the buffer is five lines in length. This window is known as the heading window.

The second window (the text window) contains 16 lines of text within the specified file. This window is 38 characters wide and 16 lines high. The white cursor (indicated by the symbol "■") will be in the upper-lefthand corner of the display awaiting editing commands.

The final five-line window found at the bottom of the display is known as the buffer window. This is used for advanced editing and is described in greater detail in the section entitled "Buffer Management."

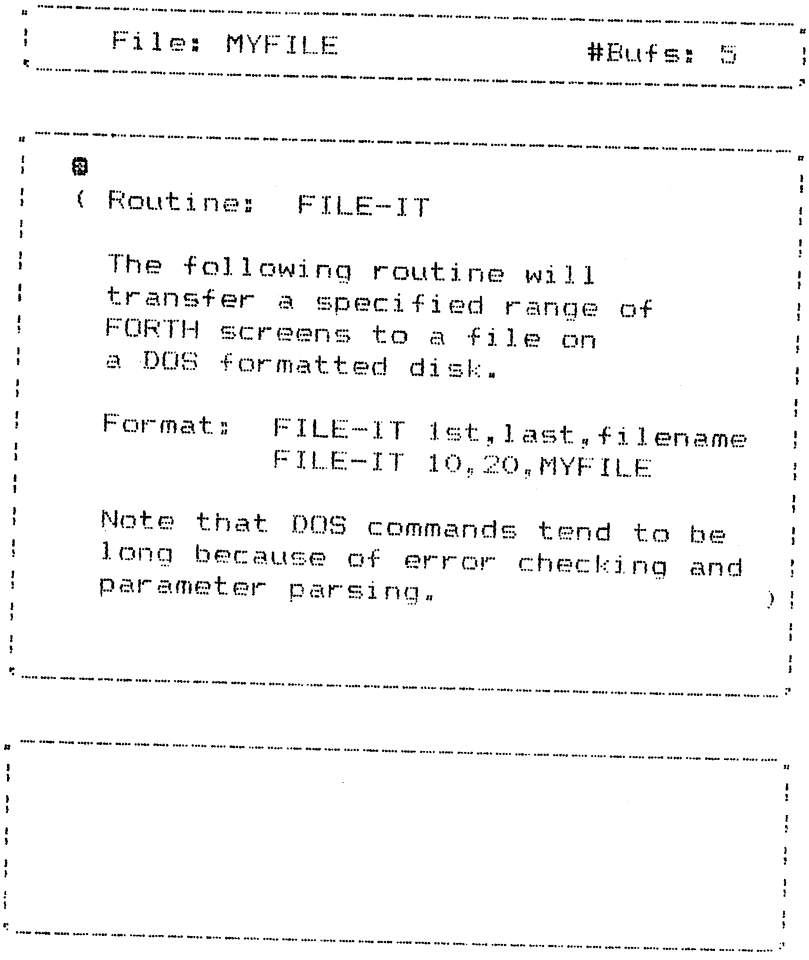


Fig. 1

LL — re-edit last file (---)

This command is used to re-edit the "Last" file edited. It functions identically to the EDIT command, except that no file names need to be specified.

Example: LL <ret> (re-edit MYFILE)

WHERE find location of error (---)

If, when compiling code, a compilation error occurs, the WHERE command will enter the edit mode and position the cursor over the last letter of the offending word. The word can then be fixed and the file saved for subsequent compilation using the "FLOAD filename/C" command.

#BUFS set buffer length (#lines ---)

The #BUFS command allows the user to specify the length (in terms of number of lines) of the special edit storage buffer. The power of the edit buffer lies in the number of lines that can be stored in it. Although the default value is five, practice shows that at least 16 lines should be set aside for this buffer. The maximum number of lines allowable is 320 which is enough to hold 20 full screens simultaneously.

.INFO display file information (---)

If an error occurs and an edited file is not saved to disk, the .INFO command will supply all the necessary information to save the file using the WRITE command.

The following sections give a detailed description of all commands which the video editor recognizes. A quick reference command list can be found following these descriptions.

Cursor Movement

When the edit mode is first entered via the EDIT command, a cursor is placed in the upper lefthand corner of the screen. It should appear as a white block and may enclose a black letter. Whenever any key is typed and it is not recognized as an editor command, it is placed in the text window where the cursor appears. Likewise, any line functions (such as delete line) work on the line where the cursor is found.

ctrl ↑ , ctrl ↓ , ctrl ← , ctrl → move-cursor commands

To change the current edit line or character, one of four commands may be given. These are known as cursor commands. They are the four keys with arrows on them. These keys move the cursor in the direction specified by the arrow on the particular key pressed. There are times, however, when this is not the case.

Similarly, if the cursor is positioned on the leftmost edge and the "cursor-left" command is given, the cursor will "wrap" to the rightmost character. Issuing "cursor-right" will wrap to the left edge.

RETURN next-line command

The RETURN key positions the cursor on the first character of the next line. If RETURN is pressed when the cursor is on the last line of the file, a line is inserted at the end of the file.

TAB tabulate command

The TAB key is used to tabulate to the next TAB stop to the right of the current cursor character.

ctrl TAB — clear TAB stop

Clear the tab stop at the current cursor location. The default tab stops can be reset by issuing the RT subcommand.

shift TAB — set TAB stop

Set a tab stop at the current cursor location. The default tab stops can be reset by issuing the RT subcommand.

ctrl L — continue search

Search for the next occurrence of the pattern set up using the PS subcommand. Patterns can be up to 30 characters in length.

ctrl V — enter subcommand mode

The puts the editor into the subcommand mode. See the section entitled "Subcommands" for a list of available commands.

NOTE:

Many commands in the editor will "mark" the file as updated so that any changes made can be preserved on disk. As simple cursor movement does not change the text window in any way, these commands never mark the file.

Editing Commands

Editing commands are those commands which modify the text in some predefined manner and mark the file as updated for later saving.

ctrl INS — character insert command

When the "insert-character" command is given, a blank character is inserted at the current cursor location. The current character and all characters to the right are pushed to the right by one character position. The last character of the line "falls off" the end and is lost. The inserted blank then becomes the current cursor character. This is the logical complement to the "delete-character" command described below.

ctrl DEL — delete character command

When the "delete-character" command is issued, the current cursor character is removed, and all characters to the right of the current cursor character are moved left one position, thus giving a "sqeeze" effect. This is normally called "closing" a line. The rightmost character on the line (which was vacated) is replaced with a blank. This serves as the logical complement to the "insert-command" described above.

shift INS — line insert command

The "line-insert" command inserts a blank line between the current cursor line and the line immediately above it. If this command is accidentally typed, the "oops" command (ctrl-O) described later can be used to recover from the mistake. Also see the "from buffer" command described in the section on buffer management for a similar command. This command serves as the logical complement to the "line-delete" command described below.

shift DEL — line delete command

The "line-delete" command deletes the current cursor line. If this command is accidentally issued, recovery can be made by issuing the "oops" command (ctrl-O) described later. Also see the "to-buffer" command described in the section on buffer management for a similar command. The "delete-line" command serves as the logical complement to the "line-insert" command.

ctrl H — erase to end of line

The "Hack" command performs a clear-to-end-of-line function. The current cursor character and all characters to the right of it on the current line are blank filled. All characters blanked are lost. The "oops" command described later can be used to recover from an accidentally hacked line.

ctrl I — insert/replace toggle

In normal operation, any key typed which is not recognized by the editor as a control command will replace the current cursor character with itself. This is the standard replace mode. Normally, if one wanted to insert a character at the current cursor location, the insert character command would have to be issued before any text could be entered. If inserting many characters, this is cumbersome.

When active, the insert submode automatically makes room for any new characters or words and frees the user from having to worry about this. When the editor is called up via the EDIT command, the insert mode is deactivated. Issuing the insert toggle command will activate it and the cursor will blink, indicating that the insert mode is on. Issuing the command a second time will deactivate the insert mode and restore the editor to the replace mode. Note that while in the insert mode, all edit commands (except BACKS, below) function as before.

BACKS — delete previous character

The BACKS key behaves in two different ways, depending upon whether the editor is in the insert mode or in the replace mode. When issued while in the replace mode, the cursor is backed up one position and the new current character is replaced with a blank. If the cursor is at the beginning of the line, the cursor does not move, but the cursor character is still replaced with a blank.

If the editor is in the insert mode, the cursor backs up one position, then deletes the new current cursor character and then closes the line. If the cursor is at the beginning of the line, the cursor remains in the same position, the cursor character is deleted and the line closed.

NOTE:

As all of the above commands modify the file in some manner, the file is marked as having been changed. This is to ensure that all changes made are eventually saved on disk. The "quit" command described later allows one to abort the edit session so that major mistakes need not be saved.

Buffer Management

Much of the utility of the file editor lies in its ability to temporarily save text in a visible buffer. To aid the user, it is possible to temporarily send text to the buffer and to later retrieve it. This storage buffer can hold as many as 320 lines of text simultaneously. This buffer is viewed through a 5 line "peephole" visible as the last window on the screen. Using this buffer, it is possible to duplicate, move, and easily reorganize text, in addition to temporarily saving a line that is about to be edited so that the original form can be viewed or restored if necessary. The following section will explain exactly how to accomplish each of these actions.

ctrl T — to buffer command

The "to-buffer" command deletes the current cursor line, but unlike the "delete-line" command where the line is lost, this command moves the "peephole" down and copies the line to the bottom line of the visible buffer window. This line is the current buffer line. The buffer is rolled upon each occurrence of this command so that it may be used repeatedly without the loss of stored text.

For example, if the cursor is positioned on ninth line of the display shown in figure 1 and the "to-buffer" command is issued twice, the final result will be as shown in figure 2.

ctrl F — from buffer command

The "from-buffer" command does exactly the opposite of the "to-buffer" command described above. It takes the current buffer line and inserts it between the current cursor line and the line above it. The cursor line and all lines below it are moved down one line. If the cursor were placed on line 14 of the above screen display and the "from-buffer" command were issued twice, the display shown in figure 3 would result.

```
┌──────────────────────────────────────────────────┐
│    File: MYFILE                  #Bufs: 5        │
└──────────────────────────────────────────────────┘

┌──────────────────────────────────────────────────┐
│                                                  │
│  ( Routine:  FILE-IT                             │
│                                                  │
│    The following routine will                    │
│    transfer a specified range of                 │
│    FORTH screens to a file on                    │
│    a DOS formatted disk.                         │
│                                                  │
current │  (                                       │
│    Note that DOS commands tend to be             │
│    long because of error checking and            │
│    parameter parsing.                          ) │
│                                                  │
│                                                  │
│  : FILE-IT DOS                ( -- )             │
│    GETARGS ?WRGARG 44 GETARG ?WRGARG             │
└──────────────────────────────────────────────────┘

┌──────────────────────────────────────────────────┐
│                                                  │
│                                                  │
│                                                  │
│                                                  │
│    Format:  FILE-IT 1st.last,filename            │
current │             FILE-IT 10,20,MYFILE         │
└──────────────────────────────────────────────────┘
```

fig. 2

```
┌──────────────────────────────────────────┐
│   File: MYFILE                #Bufs: 5   │
└──────────────────────────────────────────┘

┌──────────────────────────────────────────┐
│                                          │
│  ( Routine:  FILE-IT                     │
│                                          │
│    The following routine will            │
│    transfer a specified range of         │
│    FORTH screens to a file on            │
│    a DOS formatted disk.                 │
│                                          │
│                                          │
│    Note that DOS commands tend to be     │
│    long because of error checking and    │
│    parameter parsing.                 )  │
│                                          │
│  @ Format:  FILE-IT 1st,last,filename    │
│             FILE-IT 10,20,MYFILE         │
│  : FILE-IT DOS              ( -- )       │
└──────────────────────────────────────────┘

┌──────────────────────────────────────────┐
│   Format:  FILE-IT 1st,last,filename     │
│            FILE-IT 10,20,MYFILE          │
│                                          │
│                                          │
│                                          │
└──────────────────────────────────────────┘
```

fig. 3

ctrl K copy to buffer command

The "copy-to-buffer" command takes the current cursor line and duplicates it, sending the copy to the buffer. The cursor is then moved down one line. This commands functions identically to the "to-buffer" command described above, except that the current cursor line is NOT deleted from the text window.

ctrl U copy from buffer

The "copy-from-buffer" command replaces the current cursor line with the current buffer line. This command functions identically to the "from-buffer" command described above, except that the buffer line is not inserted into the text window, it merely replaces the current cursor line. The "oops" command described below

can be used to recover from accidental usage of this command.

ctrl R roll buffer

The "roll-buffer" command moves the buffer "peephole" down one line and redisplays the visible window. If the buffer were the minimum five lines in length, the bottom four lines in the window would move up a line and the top line would "wrap" to the bottom and become the current buffer line. If there were more than five buffer lines, the bottom four lines would move up a line, the topmost line would be pushed up behind the peephole, and a new buffer line coming up from below the peephole would be displayed and made current. For example, if the buffer were five lines long and contained:

```
          .-------------------------------------.
          ! ( Who?   )                          !
          ! ( What?  )                          !
          ! ( When?  )                          !
          ! ( Where? )                          !
Current:  ! ( Why?   )                          !
          '-------------------------------------'
```

fig. 5

the "roll-buffer" command gives:

```
          .-------------------------------------.
          ! ( What?  )                          !
          ! ( When?  )                          !
          ! ( Where? )                          !
          ! ( Why?   )                          !
Current:  ! ( Who?   )                          !
          '-------------------------------------'
```

fig. 6

ctrl B back-roll-buffer command

The "back-roll-buffer" does exactly the opposite of the "roll-buffer" command described above. For example, if given the buffer in figure 6 above, the "back-roll" command would give the buffer shown in figure 5.

ctrl C clear buffer line command

The "clear-buffer-line" command clears the current buffer line and then "back-rolls" the buffer so that successive clears can be used to erase the entire buffer.

NOTE:

Any of the above commands which change the text window will mark the file as updated. Those commands which alter only the buffer window (such as the "roll" command) will not change the status of the current screen.

Scrolling

ctrl X — previous line command

The "previous-line" command scrolls the text window up a line within the file.

ctrl E — next line command

The "next-line" command moves the text window down a line within the file.

ctrl P — previous page command

The "previous-page" command scrolls the text window up 16 lines within the file.

ctrl N — next page command

The "next-page" command moves the text window down 16 lines within the file.

ctrl S — save command

The "save" command saves any changes made to the current file and exits the edit mode.

ctrl Q — quit command

The "quit" command aborts the edit session "forgetting" any changes made to the text file in memory. The "quit" command is usually used when either the wrong file has been called up, or if it becomes desirable to start over and re-edit.

Special Commands

There are four special commands in this editor which allow greater flexibility in programming on the valFORTH system:

ESCAPE — special key command

The "special-key" command instructs the video editor to ignore any command function of the next key typed and force a character to the screen. For example, normally when "ctrl →" is typed, the cursor is moved right. By typing "ESCAPE ctrl →" the cursor is not moved -- rather, the right-arrow is displayed.

ctrl J — split line command

Often times, for formatting reasons, it is necessary to "split" a line into two lines. The split line command takes all characters to the left of the cursor and creates the first line, and with the remaining characters of the original line, a second line is created. Graphically, this looks like:

```
before:   ¦ The quick#brown fox jumped.    ¦

after:    ¦ The quick#                     ¦
          ¦           brown fox jumped.    ¦
```

ctrl G — splice (unsplit) command

The "splice-command" performs just the opposite operation of the "split-command" above.

ctrl Y — repeat command

The "repeat-command" repeats the next command or character typed until a predefined stop condition occurrs, or until a console key is pressed. This is used mostly with the previous/next page commands for continuous scrolling.

ctrl O oops command

Occassionally, a line is inserted or deleted accidentally, half a line cleared by mistake, or some other major editing blunder is made. As the name implies, the "oops" command corrects most of these major editing errors. The "oops" command can be used to recover from the following commands:

1)	insert line command	(shift INS)
2)	delete line comand	(shift DEL)
3)	hack command	(ctrl H)
4)	to buffer command	(ctrl T)
5)	from buffer command	(ctrl F)
6)	copy from buffer command	(ctrl U)
7)	split line command	(ctrl J)
8)	splice command	(ctrl G)

Subcommands

The subcommand is entered by typing ctrl-V. The display will be cleared and a prompt (": ") will be issued. The following commands may be typed in response.

ST <return> Make the current line the start of the file. (i.e., hack off the beginning)

EN <return> Make the current line the end of the file. (i.e., hack off the end)

FL <return> Position the cursor on the first line of the file.

LL <return> Position the cursor on the last line of the file.

RT <return> Reset the TAB stops to their original settings.

PS <return> Enter the pattern search submode. The user will be prompted to enter the search string. The ctrl-L command will continue the search.

IF filename <return>
Insert the specified file into the file just after the current cursor line. This is useful for pulling subroutines from another file. This can be stopped at any time by pressing a console key.

Editor Command Summary

Below is a quick reference list of all the commands which the video editor recognizes.

Entering the Edit Mode: (executed outside of the edit mode)

EDIT infile(,outfile) (---)
Enter the edit mode and edit "infile". If "outfile" is specified, send edited text to "outfile", otherwise send it "infile".

LL (---)
Re-edit the last file edited.

WHERE (---)
Enter the edit mode and position the cursor over the word that caused a compilation error.

#BUFS (#lines ---)
Sets the length (in lines) of the storage buffer. The default is five.

.INFO (---)
Display memory allocation of the current file so that it may be saved using the WRITE command. (used in case of a save error)

Cursor Movement: (issued within the edit mode)

ctrl ↑ Move cursor up one line, scrolling the file down one line if necessary.

ctrl ↓ Move cursor down one line, scrolling the file up one line if necessary.

ctrl ← Move cursor left one character, wrapping to the right edge if moved off the left.

ctrl → Move cursor right one character, wrapping to the left edge if moved off the right.

RETURN Position the cursor at the beginning of the next line. Insert line if at the end of the file.

TAB Advance to next tabular column.

ctrl TAB	Clear tab stop at current cursor location.
shift TAB	Set tab stop at current cursor location.

Editing Commands: (issued within the edit mode)

ctrl INS	Insert one blank at cursor location, losing the last character on the line.
ctrl DEL	Delete character under cursor, closing the line.
shift INS	Insert blank line above current line.
shift DEL	Delete current cursor line, closing the file.
ctrl M	Insert blank line below current line.
ctrl I	Toggle insert-mode/replace-mode. (see full description of ctrl-I).
BACKS	Delete last character typed, if on the same line as the cursor.
ctrl H	Erase to end of line (Hack).
ctrl V	Enter the subcommand mode. (see below)

Buffer Management: (issued within the edit mode)

ctrl T	Delete current cursor line sending it TO the edit buffer for later use.
ctrl F	Take the current buffer line and insert it above the current cursor line.
ctrl K	Kopy current cursor line sending it to the edit buffer for later use.
ctrl U	Take the current buffer line and copy it to the current cursor line. Unkopy
ctrl R	Roll the buffer making the next buffer line current.
ctrl B	Roll the buffer Backwards making the previous buffer line on the screen current.
ctrl C	Clear the current buffer line and performs a ctrl-B.

Note: the current buffer line is last line visible in the buffer window.

Scrolling/Saving: (issued within the edit mode)

ctrl X	Scroll the edit window up a line within the file.
ctrl E	Scroll the edit window down a line within the file.
ctrl P	Scroll the edit window up 16 lines within the file.
ctrl N	Scroll the edit window down 16 lines within the file.
ctrl S	Save the changes made to the current file and exit the edit mode.
ctrl Q	Quit the edit session forgetting all changes made to the current file.

Special Keys: (issued within the edit mode)

ESC	Do not interpret the next key typed as any of the commands above. Send it directly to the screen instead.
ctrl J	Split the current line into two lines at the point where the cursor is.
ctrl G	Splice (unsplit) the current line and the line above it.
ctrl O	Corrects any major editing blunders. Oops!
ctrl L	Continue searching for the pattern entered in the subcommand mode. Look
ctrl Y	Enter the repeat mode. The next command or character typed will be repeated until a stop condition is met, or until a console key is pressed. Used mostly with ctrl-P, ctrl-N, and the cursor commands.

Subcommands: (entered in the subcommand mode)

The subcommand is entered by typing ctrl-V. The display will be cleared and a prompt (": ") will be issued. The following commands may be typed in response.

ST <return>	Make the current line the start of the file. (i.e., hack off the beginning)

EN <return> Make the current line the end of the file. (i.e., hack off the end)

FL <return> Position the cursor on the first line of the file.

LL <return> Position the cursor on the last line of the file.

RT <return> Reset the TAB stops to their original settings.

PS <return> Enter the pattern search submode. The user will be prompted to enter the search string. The ctrl-L command will continue the search.

IF filename <return>
Insert the specified file into the file just after the current cursor line. This is useful for pulling subroutines from another file. This can be stopped at any time by pressing a console key.

valDOS I

Supplied Source Listing

LXV.

```
Screen:  10
  Ø ( valDOS:  quans and constants )
  1 '( QUAN    )( 15Ø LOAD )
  2 '( CARRAY )( 16Ø LOAD )
  3 VOCABULARY DOS IMMEDIATE
  4 DOS DEFINITIONS
  5
  6  LABEL FNFLD      11 ALLOT
  7  QUAN FNCNT       Ø TO FNCNT
  8  QUAN FNSEP       Ø TO FNSEP
  9  QUAN FNSWCH      Ø TO FNSWCH
 1Ø  QUAN UNIT        Ø TO UNIT
 11  QUAN DFLUNT      Ø TO DFLUNT
 12  QUAN FNSEC       Ø TO FNSEC
 13  QUAN ?WILD       Ø TO ?WILD
 14  QUAN #ENTRIES    Ø TO #ENTRIES
 15  QUAN #FREE       Ø TO #FREE  -->
```

```
Screen:  11
  Ø ( valDOS:  quans and constants )
  1
  2  QUAN DSKERR      Ø TO DSKERR
  3  QUAN DIRBLK      Ø TO DIRBLK
  4  QUAN FSMBLK      Ø TO FSMBLK
  5  QUAN ?FSMUP      Ø TO ?FSMUP
  6  QUAN ?SAME
  7  QUAN DIRFRE
  8  QUAN FL#
  9  QUAN TADR
 1Ø  QUAN #UNTRN
 11  QUAN N1$
 12  QUAN N2$
 13  QUAN DIRLOC
 14  QUAN WRKSPC
 15  QUAN DOSTMP                 -->
```

```
Screen:  12
  Ø ( valDOS:  quans and constants )
  1
  2 VECT RDMOPN
  3 ASSIGN NOOP TO RDMOPN
  4
  5 VECT RDMCLS
  6 ASSIGN NOOP TO RDMCLS
  7
  8 VECT RDMSPC
  9 ASSIGN NOOP TO RDMSPC
 1Ø
 11 VECT RDMENDF
 12 ASSIGN NOOP TO RDMENDF
 13
 14 VECT RDMWND
 15 ASSIGN NOOP TO RDMWND        -->
```

```
Screen:  13
  Ø ( valDOS:  arrays                )
  1
  2   4 CONSTANT MAXFL
  3 128 CONSTANT 128
  4
  5 ( addresses of file buffers )
  6 MAXFL 1+ ARRAY FLBUF
  7 Ø FLBUF MAXFL 1+ 2* ERASE
  8
  9 ( map of buffers in use )
 1Ø MAXFL 2* 2 + ARRAY OPNSTT
 11 Ø OPNSTT MAXFL 4 * 2 + ERASE
 12
 13 ( the filename "Dx:*" )
 14 LABEL ALLNMS
 15 Ø4 C, 68 C, Ø C, 58 C, 42 C, -->
```

```
Screen:  14
  Ø ( valDOS:  error codes           )
  1
  2  1 CONSTANT BADNME
  3  2 CONSTANT BADUNT
  4  3 CONSTANT BADFSM
  5  4 CONSTANT FLEXST
  6  5 CONSTANT DIRFUL
  7  6 CONSTANT DSKFUL
  8  7 CONSTANT AMBNME
  9  8 CONSTANT FLDNE
 1Ø  9 CONSTANT TMFOPN
 11 1Ø CONSTANT EOFERR
 12 11 CONSTANT FLNOPN
 13 12 CONSTANT BADFL#
 14 13 CONSTANT FLWPRT
 15 14 CONSTANT WRGARG              -->
```

```
Screen:  15
  Ø ( valDOS:  error codes           )
  1 15 CONSTANT FLOPN
  2 16 CONSTANT FLTBG
  3
  4 HERE 1 AND ALLOT  ( even addr )
  5 144 MAXFL * 192 + ALLOT
  6 HERE TO WRKSPC 128 ALLOT
  7 : TOPOM 741 @ 1- ;
  8
  9 : FSMAP                  ( -- a )
 1Ø   WRKSPC 128 - ;
 11
 12 : DIRTBL FSMAP 64 - ;    ( -- a )
 13
 14 : BUFBOT                ( -- a )
 15   DIRTBL 144 MAXFL * - ;     -->
```

```
Screen:  16
  0 ( valDOS:  error routines       )
  1
  2 : DOSERR                    ( # -- )
  3   TO DSKERR R> DROP 0 ;
  4
  5 : ?DOSERR                ( f # -- )
  6   SWAP 0=
  7   IF
  8     R> DROP DOSERR
  9   ENDIF
 10   DROP ;
 11
 12 : ?DSKERR                   ( f -- )
 13   0= IF R> DROP
 14   DSKERR DOSERR ENDIF ;
 15                                  -->
```

```
Screen:  17
  0 ( valDOS:  filename conversion )
  1
  2 : GETSWCH           ( a c -- a c )
  3   0 TO FNSWCH DUP 2 >
  4   IF 2DUP + 2- C@ 47 ( "/" ) =
  5     IF
  6       2- 2DUP + 1+ C@
  7       TO FNSWCH
  8     ENDIF
  9   ENDIF ;
 10
 11 : GETUNIT           ( a c -- a c )
 12   DFLUNT TO UNIT DUP 2 >
 13   IF OVER @ 14916 ( "D:") =
 14    IF
 15      2- SWAP 2+ SWAP             -->
```

```
Screen:  18
  0 ( valDOS:  filename conversion )
  1
  2    ELSE DUP 3 >
  3     IF OVER C@ 68 ( "D") =
  4      3 PICK 2+ C@ 58 = AND
  5      IF OVER 1+ C@ 49 ( "1") -
  6      DUP 0< NOT OVER 3 > NOT AND
  7       IF
  8        TO UNIT 3 - SWAP 3 + SWAP
  9       ELSE
 10        2DROP DROP BADUNT DOSERR
 11       ENDIF
 12      ENDIF
 13     ENDIF
 14    ENDIF
 15   ENDIF 1 ;                      -->
```

```
Screen:  19
  0 ( valDOS:  filename conversion )
  1
  2 : FNCON                     ( $ -- f )
  3   0 TO DSKERR
  4   COUNT GETSWCH GETUNIT ?DSKERR
  5   8 TO FNCNT 46 TO FNSEP
  6   FNFLD 11 BLANKS FNFLD
  7   BEGIN
  8     3 PICK C@ DUP FNSEP =
  9     IF
 10       2DROP FNFLD 7 +
 11       DUP 1+ 3 BLANKS
 12       3 TO FNCNT 0 TO FNSEP
 13     ELSE
 14       DUP 42 ( "*" ) =
 15                                  -->
```

```
Screen:  20
  0 ( valDOS:  filename conversion )
  1
  2       IF
  3         OVER FNFLD 11 + OVER -
  4         63 ( "?") FILL DROP
  5       ELSE
  6         DUP 48 ( "0") <  OVER
  7         90 ( "Z") > OR   OVER
  8         57 ( "9") >    3 PICK
  9         65 ( "A") < AND OR
 10         OVER 63 ( "?") <> AND
 11         IF 2DROP 2DROP
 12           BADNME DOSERR
 13         ENDIF
 14         OVER C!
 15       ENDIF                      -->
```

```
Screen:  21
  0 ( valDOS:  filename conversion )
  1       FNCNT 1- DUP TO FNCNT 0<
  2       IF DROP 2DROP
  3          BADNME DOSERR
  4       ENDIF
  5     ENDIF
  6     1+ ROT 1+ ROT 1- ROT
  7     OVER 0=
  8   UNTIL
  9   2DROP DROP FNFLD C@ 63 >=
 10   BADNME ?DOSERR 0 TO ?WILD
 11   11 0
 12   DO
 13     FNFLD I + C@ 63 =
 14     IF 1 TO ?WILD ENDIF
 15   LOOP 1 ;                       -->
```

```
Screen:  22
  Ø ( valDOS:  alias definitions   )
  1
  2 : 3PICK                    ( -- n )
  3   3 PICK ;
  4
  5 : FLBUF@                 ( n -- a )
  6   FLBUF @ ;
  7
  8 : FLINFO                   ( -- a )
  9   FL# FLBUF@ ;
 1Ø
 11 : 72Ø* 72Ø * ;
 12 : 256* 256 * ;
 13 : 128- 128 - ;
 14 : 135- 135 - ;
 15 : 4* 2* 2* ;                   -->
```

```
Screen:  23
  Ø ( valDOS:  get & put directory )
  1
  2 : DIRGET              ( sector -- )
  3   OFFSET @ OVER DRØ BLOCK
  4   TO DIRLOC OFFSET ! TO DIRBLK ;
  5
  6 : DIRFLS                   ( -- )
  7   FLUSH Ø TO DIRBLK ;
  8
  9 : DIRUP                    ( -- )
 1Ø   UPDATE ;
 11
 12 : ENTRY              ( unit # -- )
 13   1- 8 /MOD 36Ø +
 14   ROT 72Ø* + DIRGET
 15   16 * DIRLOC + ;              -->
```

```
Screen:  24
  Ø ( valDOS:  get free space map  )
  1
  2 : FSMGET               ( unit -- )
  3   72Ø* 359 +
  4   DUP FSMBLK <>
  5   IF
  6     ?FSMUP
  7     IF
  8       FSMAP FSMBLK Ø R/W
  9     ENDIF
 1Ø     FSMAP OVER 1 R/W
 11     TO FSMBLK Ø TO ?FSMUP
 12   ELSE
 13     DROP
 14   ENDIF ;
 15                                -->
```

```
Screen:  25
  Ø ( valDOS:  put free space map  )
  1
  2 : FSMFLS                   ( -- )
  3   ?FSMUP
  4   IF
  5     FSMAP FSMBLK Ø R/W
  6   ENDIF
  7   Ø TO ?FSMUP
  8   Ø TO FSMBLK ;
  9
 1Ø : FSMUP                    ( -- )
 11   1 TO ?FSMUP ;
 12
 13 : DSKFLS                   ( -- )
 14   DIRFLS FSMFLS ;
 15                                -->
```

```
Screen:  26
  Ø ( valDOS:  check directory     )
  1
  2 : CHKDIR                 ( -- # )
  3   Ø TO #ENTRIES Ø TO #FREE
  4   65 TO DIRFRE Ø 368 36Ø
  5   DO
  6    I UNIT 72Ø* + DIRGET
  7    DIRLOC 128 +•DIRLOC
  8    DO
  9     1+ 1 TO ?SAME
 1Ø     I C@ 195 AND 66 =
 11     IF 11 Ø
 12      DO
 13       J I + 5 + C@ FNFLD I +
 14       C@ DUP 63 ( "?") =
 15                                -->
```

```
Screen:  27
  Ø ( valDOS:  check directory     )
  1
  2        IF DROP DUP ENDIF <>
  3        IF Ø TO ?SAME LEAVE ENDIF
  4      LOOP ?SAME
  5      IF
  6        DUP #ENTRIES DIRTBL + C!
  7        1 AT #ENTRIES +!
  8      ENDIF
  9     ELSE
 1Ø      DUP DIRFRE MIN TO DIRFRE
 11      1 AT #FREE +!
 12     ENDIF
 13    16 /LOOP
 14   LOOP
 15   DROP #ENTRIES ;              -->
```

```
Screen:  28
  Ø ( valDOS:  allocate a sector    )
  1
  2 : ALTSEC        ( unit - [# t]/f )
  3   FSMGET
  4   FSMAP 3 + @ DUP
  5   IF
  6    1- FSMAP 3 + !
  7    Ø -1 FSMAP 1Ø + 9Ø Ø+S
  8    DO
  9     I C@
 1Ø     IF
 11      SWAP DROP DUP LEAVE
 12      128 I C@
 13      BEGIN DUP 128 AND Ø=
 14      WHILE 2* ROT 1+ ROT 2/ ROT
 15      REPEAT                    -->
```

```
Screen:  29
  Ø ( valDOS:  allocate a sector    )
  1
  2       DROP I SWAP TOGGLE
  3     ELSE
  4       8 +
  5     ENDIF
  6    1 /LOOP
  7    SWAP Ø=
  8    IF
  9      DROP BADFSM DOSERR
 1Ø    ENDIF
 11    FSMUP 1
 12   ENDIF ;
 13
 14
 15                             -->
```

```
Screen:  3Ø
  Ø ( valDOS:  release a sector     )
  1
  2 : RELSEC              ( unit # -- )
  3   SWAP FSMGET
  4   1+ 8 /MOD FSMAP 1Ø + +
  5   SWAP 128
  6   BEGIN
  7     OVER
  8   WHILE
  9     2/ SWAP 1- SWAP
 1Ø   REPEAT
 11   SWAP DROP OVER C@
 12   OR SWAP C!
 13   FSMAP 3 + DUP @ 1+ SWAP !
 14   FSMUP ;
 15                             -->
```

```
Screen:  31
  Ø ( valDOS:  find free buffer     )
  1
  2 : NXTOPN                  ( -- a )
  3   MAXFL 4* Ø
  4   DO
  5    I -144 * DIRTBL 16 - + I' Ø
  6    DO
  7      I OPNSTT @ OVER =
  8      IF DROP Ø LEAVE ENDIF
  9    LOOP -DUP
 1Ø    IF DUP I' Ø
 11      DO I OPNSTT @ Ø=
 12        IF I OPNSTT ! LEAVE ENDIF
 13      LOOP LEAVE
 14    ENDIF
 15   LOOP ;                        -->
```

```
Screen:  32
  Ø ( valDOS:  flush file buffer    )
  1
  2 : FLFLS                 ( fl# -- )
  3   FLBUF@ DUP 1Ø + C@
  4   IF
  5     DUP 8 + C@ 72Ø*
  6     OVER 5 + @ +
  7     OVER 128- SWAP Ø R/W
  8     1Ø + Ø SWAP C!
  9   ELSE
 1Ø     DROP
 11   ENDIF ;
 12
 13
 14
 15                             -->
```

```
Screen:  33
  Ø ( valDOS:  [OPEN]                )
  1
  2 : (OPEN)       ( $ -- [a # 1]/Ø )
  3   FNCON ?DSKERR
  4   ?WILD NOT AMBNME ?DOSERR
  5   CHKDIR FLDNE ?DOSERR
  6   Ø MAXFL 1+ 1
  7   DO
  8     I FLBUF@ Ø=
  9     IF I + LEAVE ENDIF
 1Ø   LOOP
 11   -DUP TMFOPN ?DOSERR
 12   >R NXTOPN DUP R FLBUF !
 13   DUP 16 ERASE
 14   UNIT DIRTBL C@ ENTRY
 15                             -->
```

```
Screen:  34
  Ø ( valDOS:  [OPEN]                    )
  1
  2   OVER 5 CMOVE
  3   3 + DUP @ 1- OVER 2+ !
  4   5 + UNIT OVER C! 1+
  5   DIRTBL C@ SWAP C!
  6   R FLBUF@ 128- DUP
  7   UNIT 72Ø* OVER 133 +
  8   @ + 1 R/W R>
  9   RDMOPN 1 ;
 1Ø
 11
 12
 13
 14
 15                                   -->
```

```
Screen:  35
  Ø ( valDOS:  [CLOSE]                   )
  1
  2 : (CLOSE)                  ( fl# -- )
  3   -DUP
  4   IF DUP ELSE MAXFL 1+ 1 ENDIF
  5   DO
  6    I DUP TO FL# FLBUF@
  7    IF
  8      RDMCLS FL# FLFLS
  9      FLINFO 2+ C@ 128 AND
 1Ø      IF
 11        FLINFO DUP 8 + C@
 12        OVER 9 + C@ ENTRY 1+
 13        SWAP 1+ @ 32767 AND
 14        SWAP ! DIRUP DIRFLS
 15      ENDIF                        -->
```

```
Screen:  36
  Ø ( valDOS:  [CLOSE]                   )
  1
  2       FLINFO Ø FL# FLBUF !
  3       Ø OPNSTT
  4       BEGIN
  5         DUP @ 3PICK <>
  6       WHILE
  7         2+
  8       REPEAT
  9       SWAP DROP Ø SWAP !
 1Ø     ENDIF
 11   LOOP ;
 12
 13
 14
 15                                   -->
```

```
Screen:  37
  Ø ( valDOS:  [READ]                    )
  1
  2 : (READ)     ( adr cnt fl# -- f )
  3   TO FL# TO #UNTRN TO TADR
  4   FL# 1 < FL# MAXFL > OR
  5   NOT BADFL# ?DOSERR
  6   FLINFO FLNOPN ?DOSERR
  7   BEGIN
  8     FLINFO 7 + DUP C@
  9     OVER 8 - C@ =
 1Ø     IF
 11      DUP 1Ø - DUP C@ 3 AND
 12      256* SWAP 1+ C@ + -DUP Ø=
 13      IF DROP EOFERR DOSERR ENDIF
 14      1- FL# FLFLS OVER 135-
 15      OVER 4 PICK 1+ C@            -->
```

```
Screen:  38
  Ø ( valDOS:  [READ]                    )
  1
  2       72Ø* + 1 R/W
  3       OVER 2- ! Ø OVER C!
  4     ENDIF
  5     DUP 8 - C@ OVER C@ -
  6     #UNTRN OVER U<
  7     IF DROP #UNTRN ENDIF
  8     DUP 3PICK 7 + DUP @
  9     ROT + SWAP !
 1Ø     OVER 135- 3PICK C@ +
 11     TADR DUP 4 PICK + TO TADR
 12     3PICK CMOVE #UNTRN OVER -
 13     TO #UNTRN OVER C@ +
 14     SWAP C! #UNTRN Ø=
 15   UNTIL 1 ;                       -->
```

```
Screen:  39
  Ø ( valDOS:  [WRITE]                   )
  1
  2 : (WRITE)    ( adr cnt fl# -- f )
  3   TO FL# TO #UNTRN TO TADR
  4   FL# 1 < FL# MAXFL > OR
  5   NOT BADFL# ?DOSERR
  6   FLINFO -DUP FLNOPN ?DOSERR
  7   C@ 32 AND NOT FLWPRT ?DOSERR
  8   BEGIN
  9     FLINFO 7 + DUP C@ 125 =
 1Ø     IF
 11       DUP 1Ø - DUP
 12       C@ 3 AND 256* SWAP 1+
 13       C@ + -DUP Ø=
 14       IF DUP 1+ C@ ALTSEC Ø=
 15                                   -->
```

```
Screen:  40
  0 ( valDOS:  [WRITE]                )
  1
  2          IF
  3            DROP DSKFUL DOSERR
  4          ENDIF
  5          DUP 1+ 255 AND
  6          OVER 1+ 32767 AND 256
  7          / 4 PICK 2+ C@ 1- 4*
  8          >R R OR 4 PICK 10 - C!
  9          3PICK 9 - C! 1 3PICK
 10          3 + C! FL# FLFLS OVER
 11          135- 128 ERASE R>
 12          3PICK 10 - C! OVER 6 -
 13          DUP @ 1+ 32768 OR SWAP !
 14
 15                                  -->
```

```
Screen:  41
  0 ( valDOS:  [WRITE]                )
  1
  2        ELSE
  3          FL# FLFLS 1- OVER
  4          135- OVER 4 PICK
  5          1+ C@ 720* + 1 R/W
  6        ENDIF
  7        OVER 2- ! 0 OVER C!
  8      ENDIF
  9      125 OVER C@ -
 10      #UNTRN OVER U<
 11      IF DROP #UNTRN ENDIF
 12      DUP 3PICK 7 + DUP
 13      @ ROT + SWAP !
 14      OVER 135- 3PICK C@ +
 15                                  -->
```

```
Screen:  42
  0 ( valDOS:  [WRITE]  [WRB]         )
  1
  2      TADR DUP 4 PICK + TO TADR
  3      SWAP 3PICK CMOVE
  4      OVER C@ OVER + DUP
  5      4 PICK C! 3PICK 8 -
  6      DUP C@ ROT MAX SWAP C!
  7      #UNTRN SWAP - TO #UNTRN
  8      1 SWAP 3 + C!
  9      #UNTRN 0=
 10    UNTIL 1 ;
 11
 12 : (WRB)              ( b f1# -- f )
 13    SWAP WRKSPC C!
 14    WRKSPC 1 ROT (WRITE) ;
 15                                  -->
```

```
Screen:  43
  0 ( valDOS:  [RDB]                  )
  1
  2 : (RDB)          ( f1# -- [b 1]/0 )
  3    DUP TO FL#
  4    1 < FL# MAXFL > OR
  5    NOT BADFL# ?DOSERR
  6    FLINFO FLNOPN ?DOSERR
  7    FLINFO 7 +
  8    DUP C@ OVER 8 - C@ =
  9    IF
 10      DUP 10 - DUP
 11      C@ 3 AND 256*
 12      SWAP 1+ C@ + -DUP 0=
 13      IF
 14        DROP EOFERR DOSERR
 15      ENDIF                       -->
```

```
Screen:  44
  0 ( valDOS:  [RDB]                  )
  1
  2      1- FL# FLFLS
  3      OVER 135- OVER
  4      4 PICK 1+ C@ 720* +
  5      1 R/W
  6      OVER 2- ! 0 OVER C!
  7    ENDIF
  8    DUP C@ SWAP
  9    OVER 1+ OVER C!
 10    1 OVER 7 + +!
 11    135- + C@ 1 ;
 12
 13
 14 FORTH DEFINITIONS
 15
```

```
Screen:  45
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen:  46
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen:  47
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen:  48
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen:  49
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen:  50
 0 ( valDOS:  [WIND]                    )
 1 '( DOS DOS )(  )
 2 '( FNFLD )( 10 LOAD )
 3
 4 DOS DEFINITIONS
 5
 6 : (WIND)              ( f fl# -- f )
 7   TO FL# TO DOSTMP
 8   FL# 1 < FL# MAXFL > OR
 9   NOT BADFL# ?DOSERR RDMWND
10   FLINFO -DUP FLNOPN ?DOSERR
11   DOSTMP
12   IF ( rewinding )
13     FL# FLFLS DUP 3 + @ 1-
14     DUP 3PICK 5 + @ <>
15                                  -->
```

```
Screen:  51
 0 ( valDOS:  [WIND]                    )
 1
 2     IF
 3       DUP 3PICK 5 + ! OVER 8 +
 4       C@ 720* + OVER 128-
 5       SWAP 1 R/W
 6     ELSE DROP ENDIF
 7     0 OVER 7 + C!
 8     0 SWAP 14 + !
 9   ELSE  ( to end of file )
10     DROP
11     BEGIN
12       WRKSPC 128 FL# (READ) 0=
13     UNTIL
14     DSKERR EOFERR = ?DSKERR
15   ENDIF 1 ;                         -->
```

```
Screen:  52
  Ø ( valDOS:  [ENTER]                  )
  1
  2 : (ENTER)                  ( $ -- f )
  3   FNCON ?DSKERR
  4   ?WILD NOT AMBNME ?DOSERR
  5   CHKDIR Ø= FLEXST ?DOSERR
  6   #FREE DIRFUL ?DOSERR
  7   UNIT ALTSEC DSKFUL ?DOSERR
  8   DUP 1+ TO FNSEC
  9   WRKSPC 128 ERASE
 1Ø   DIRFRE 1- 4* WRKSPC
 11   125 + C! WRKSPC
 12   SWAP UNIT 72Ø* + Ø R/W
 13   UNIT DIRFRE ENTRY
 14   66 OVER C! 1+ 1 OVER ! 2+
 15                              -->
```

```
Screen:  53
  Ø ( valDOS:  [ENTER]                  )
  1
  2   FNSEC OVER ! 2+
  3   FNFLD SWAP 11 CMOVE
  4   DIRUP FSMUP 1 ;
  5
  6
  7
  8
  9
 1Ø
 11
 12
 13
 14
 15                              -->
```

```
Screen:  54
  Ø ( valDOS:  [SPACE]                  )
  1
  2 : (SPACE)          ( cnt fl# -- f )
  3   TO FL#
  4   FL# 1 < FL# MAXFL > OR
  5   IF DROP BADFL# DOSERR ENDIF
  6   FLINFO Ø=
  7   IF DROP FLNOPN DOSERR ENDIF
  8   RDMSPC DUP Ø<   ( backwards )
  9   IF
 1Ø     FLINFO 7 + DUP C@
 11     3PICK + DUP Ø<
 12     IF
 13       DROP 7 + @ +
 14       Ø MAX 1 FL# (WIND) Ø=
 15                              -->
```

```
Screen:  55
  Ø ( valDOS:  [SPACE]                  )
  1
  2       IF
  3         DROP DSKERR DOSERR
  4       ENDIF
  5     ELSE
  6       ROT DROP SWAP DUP
  7       C@ MINUS OVER 7 + +!
  8       Ø SWAP C!
  9     ENDIF
 1Ø   ENDIF
 11   128 /MOD
 12   BEGIN DUP    ( space forward )
 13   WHILE
 14     WRKSPC 128 FL# (READ) Ø=
 15                              -->
```

```
Screen:  56
  Ø ( valDOS:  [SPACE]                  )
  1
  2     IF
  3       2DROP DSKERR DOSERR
  4     ENDIF
  5     1-
  6   REPEAT
  7   DROP WRKSPC SWAP
  8   FL# (READ) ?DSKERR
  9   1 ;
 1Ø
 11
 12
 13
 14
 15                              -->
```

```
Screen:  57
  Ø ( valDOS:  [?OPEN]                  )
  1
  2 : (?OPEN)          ( $ - [f 1]/Ø )
  3   FNCON ?DSKERR
  4   ?WILD NOT AMBNME ?DOSERR
  5   CHKDIR FLDNE ?DOSERR
  6   UNIT DIRTBL C@ 256* +
  7   Ø MAXFL 1+ 1
  8   DO
  9     I FLBUF@ -DUP
 1Ø     IF 8 + @ 3PICK =
 11       IF 1+ LEAVE ENDIF
 12     ENDIF
 13   LOOP
 14   SWAP DROP 1 ;
 15                              -->
```

```
Screen:  58
  Ø ( valDOS:  [REN]                  )
  1
  2 : (REN)              ( $n $o -- f )
  3   TO N1$ TO N2$
  4   N2$ FNCON ?DSKERR
  5   ?WILD NOT AMBNME ?DOSERR
  6   CHKDIR Ø= FLEXST ?DOSERR
  7   FNFLD WRKSPC 11 CMOVE
  8   UNIT TO DOSTMP
  9   N1$ FNCON ?DSKERR
 1Ø   ?WILD NOT AMBNME ?DOSERR
 11   DOSTMP UNIT = BADUNT ?DOSERR
 12   CHKDIR FLDNE ?DOSERR
 13   UNIT DIRTBL C@ ENTRY C@
 14   32 AND Ø= FLWPRT ?DOSERR
 15                              -->
```

```
Screen:  59
  Ø ( valDOS:  [REN]  [KILL]          )
  1
  2   WRKSPC UNIT
  3   DIRTBL C@ ENTRY 5 +
  4   11 CMOVE DIRUP 1 ;
  5
  6 : (KILL)                  ( $ -- f )
  7   (?OPEN) ?DSKERR
  8   NOT FLOPN ?DOSERR
  9   UNIT DIRTBL C@ ENTRY
 1Ø   DUP C@ 32 AND
 11   IF DROP FLWPRT DOSERR ENDIF
 12   DUP 3 + @ 1-
 13   BEGIN
 14     WRKSPC OVER
 15     UNIT 72Ø* + 1 R/W        -->
```

```
Screen:  6Ø
  Ø ( valDOS:  [KILL]                 )
  1
  2     UNIT SWAP RELSEC
  3     WRKSPC 125 + DUP
  4     1+ C@ DUP ROT C@
  5     3 AND DUP <ROT
  6     256* + 1- <ROT OR Ø=
  7   UNTIL
  8   DROP 128 SWAP C!
  9   DIRUP 1 ;
 1Ø
 11
 12
 13
 14
 15                              -->
```

```
Screen:  61
  Ø ( valDOS:  [LOCK]  [UNLOCK]       )
  1
  2 : (DOLCK)              ( $ opt -- f )
  3   TO DOSTMP (?OPEN) ?DSKERR
  4   NOT FLOPN ?DOSERR
  5   DOSTMP UNIT DIRTBL C@
  6   ENTRY DUP C@ ROT
  7   IF 32 OR ELSE 223 AND ENDIF
  8   SWAP C! DIRUP 1 ;
  9
 1Ø : (LOCK)                   ( $ -- f )
 11   1 (DOLCK) ?DSKERR 1 ;
 12
 13 : (UNLOCK)                 ( $ -- f )
 14   Ø (DOLCK) ?DSKERR 1 ;
 15                              -->
```

```
Screen:  62
  Ø ( valDOS:  [ENDF]                 )
  1
  2 : (ENDF)                 ( fl# -- f )
  3   TO FL#
  4   FL# 1 < FL# MAXFL > OR
  5   NOT BADFL# ?DOSERR
  6   FLINFO -DUP FLNOPN ?DOSERR
  7   DUP C@ 32 AND
  8   IF DROP FLWPRT DOSERR ENDIF
  9   DUP 7 + C@ OVER 1- C!
 1Ø   DUP 2- DUP C@ OVER 1- C@
 11   3 AND ROT DUP 11 + C@
 12   1- 4* SWAP 1- ! 1 4
 13   PICK 1Ø + C!
 14   ROT 8 + C@ <ROT
 15                              -->
```

```
Screen:  63
  Ø ( valDOS:  [ENDF]                 )
  1
  2   BEGIN
  3     2DUP OR
  4   WHILE
  5     256* + 1- OVER 72Ø*
  6     OVER + WRKSPC SWAP 1 R/W
  7     OVER SWAP RELSEC
  8     WRKSPC 126 + DUP C@
  9     SWAP 1- C@ 3 AND
 1Ø     FLINFO 1+ DUP @
 11     1- 32768 OR SWAP !
 12   REPEAT
 13   2DROP DROP RDMENDF 1 ;
 14
 15 FORTH DEFINITIONS
```

Screen: 64
Ø
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 65
Ø
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 66
Ø
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 67
Ø
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 68
Ø
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 69
Ø
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```
Screen:  70
 Ø ( valDOS:  quans and vects      )
 1 '( DOS DOS )(  )
 2 '( (WIND) )( 5Ø LOAD )
 3 '( $. )( : $. COUNT TYPE ; )
 4 '( +Y/N )( 142 LOAD )
 5
 6   QUAN FLFL#  QUAN FLCNT
 7   QUAN ED#IN  Ø TO ED#IN
 8   LABEL FLNME 16 ALLOT
 9   VECT *ECHO
1Ø   ASSIGN DROP TO *ECHO
11
12 : ECHO                    ( f -- )
13   IF ASSIGN EMIT
14   ELSE ASSIGN DROP
15   ENDIF TO *ECHO ;              -->
```

```
Screen:  71
 Ø ( valDOS:  argument evaluation )
 1
 2 : GETVAL DOS           ( $ -- v )
 3   WRKSPC 34 BLANKS
 4   WRKSPC OVER C@ 1+ CMOVE
 5   WRKSPC LATEST (FIND)
 6   IF
 7     DROP CFA EXECUTE
 8   ELSE
 9     WRKSPC NUMBER DROP
1Ø   ENDIF ;
11
12 : GETARGS                 ( -- $ )
13   32 WORD HERE 1 OVER @ 1 =
14   IF 2DROP Ø ENDIF Ø 84 C@
15   4Ø * 85 @ + 88 @ + C! ;     -->
```

```
Screen:  72
 Ø ( valDOS:  argument evaluation )
 1
 2 : GETARG   ( delm -- $ [$ 1]/Ø )
 3   DOS OVER COUNT
 4   BEGIN
 5     DUP Ø# 3PICK C@
 6     5 PICK <> AND
 7   WHILE 1- SWAP 1+ SWAP
 8   REPEAT
 9   ROT DROP -DUP
1Ø   IF
11     DUP 1- 3PICK C! ROT
12     DUP C@ ROT - OVER C! 1
13   ELSE
14   DROP Ø ENDIF ;
15                               -->
```

```
Screen:  73
 Ø ( valDOS:  error routines       )
 1
 2 : CMDERR DOS             ( err -- )
 3   CR DUP TO DSKERR 4 /MOD
 4   696 + WRKSPC SWAP 1 R/W
 5   32 * WRKSPC + 32 -TRAILING
 6   TYPE CR SP! DSKFLS QUIT ;
 7
 8 : ?CMDERR              ( f err -- )
 9   SWAP Ø= IF CMDERR ENDIF DROP ;
1Ø
11 : ?WRGARG DOS             ( f -- )
12   WRGARG ?CMDERR ;
13
14 : ?SYSERR DOS             ( f -- )
15   DSKERR ?CMDERR ;              -->
```

```
Screen:  74
 Ø ( valDOS:  CONFN                )
 1
 2 LABEL FNAME 16 ALLOT
 3
 4 : CONFN DOS           ( a u -- $ )
 5   FNAME 1+ 68 OVER C!
 6   1+ SWAP 49 + OVER C!
 7   1+ 58 OVER C!
 8   1+ OVER 8 -TRAILING
 9   >R OVER R CMOVE R> +
1Ø   46 OVER C! 1+ SWAP 8 +
11   3 -TRAILING  >R OVER R
12   CMOVE R + R> Ø= -
13   FNAME 1+ - FNAME
14   >R R C! R> ;
15                               -->
```

```
Screen:  75
 Ø ( valDOS:  FLOAD support        )
 1
 2 : (SKIP) DOS          ( n fl# -- )
 3   FNSWCH 67 ( "C") =
 4   IF
 5     ED#IN OVER (SPACE) ?SYSERR
 6   ENDIF
 7   SWAP
 8   BEGIN -DUP
 9   WHILE 1-
1Ø     BEGIN
11       OVER (RDB) ?SYSERR
12       155 =
13     UNTIL
14   REPEAT
15   DROP ;                       -->
```

```
Screen:  76
  Ø ( valDOS:  FLOAD support          )
  1
  2 : NXTCHR DOS          ( -- [c 1]/Ø )
  3   ?TERMINAL Ø= STATE @ Ø#
  4   OR EOFERR ?DOSERR
  5   FLFL# (RDB) ?DSKERR
  6   1 AT FLCNT +!  DUP *ECHO
  7   DUP 155 =
  8   IF
  9     DROP BL
 1Ø     FLCNT Ø TO FLCNT 39 =
 11     IF
 12       DROP [ LATEST PFA CFA , ]
 13       ?DSKERR
 14     ENDIF
 15   ENDIF 1 ;                      -->
```

```
Screen:  77
  Ø ( valDOS:  FLOAD support          )
  1
  2 : FWORD DOS                ( c -- )
  3   HERE 34 BLANKS HERE 1+ DUP
  4   BEGIN
  5     DROP NXTCHR DUP
  6     DSKERR EOFERR = OR ?SYSERR
  7     Ø= IF Ø ENDIF
  8     DUP 4 PICK <>
  9   UNTIL
 1Ø   BEGIN
 11     OVER C! 1+
 12     TOPOM 32 - OVER U>
 13     FLTBG ?CMDERR
 14     NXTCHR DUP
 15                                  -->
```

```
Screen:  78
  Ø ( valDOS:  FLOAD                  )
  1
  2     DSKERR EOFERR = OR ?SYSERR
  3     Ø= IF Ø ENDIF
  4     DUP 4 PICK = OVER Ø= OR
  5   UNTIL
  6   ROT 2DROP
  7   HERE 1+ - HERE C! ;
  8
  9 : FLOAD DOS                  ( -- )
 1Ø   GETARGS ?WRGARG 44 GETARG
 11   IF SWAP GETVAL 1- Ø MAX
 12   ELSE Ø ENDIF
 13   SWAP FLFL# >R
 14   ' WORD DUP @ >R 2+ @ >R
 15   [ ' BRANCH CFA ] LITERAL     -->
```

```
Screen:  79
  Ø ( valDOS:  FLOAD                  )
  1
  2   ' WORD ! ' FWORD ' WORD 2+ -
  3   ' WORD 2+ !  BLK @ >R
  4   FLCNT >R Ø BLK !
  5   DUP FLNME 16 CMOVE
  6   (OPEN) ?SYSERR
  7   TO FLFL# DROP
  8   FLFL# (SKIP) CR INTERPRET
  9   FLFL# (CLOSE) DIRFLS
 1Ø   R> TO FLCNT R> BLK !
 11   ' WORD R> OVER
 12   2+ ! R> SWAP !
 13   R> TO FLFL# ;
 14
 15                                   -->
```

```
Screen:  8Ø
  Ø ( valDOS:  LOAD redefined         )
  1
  2 : FIXWORD DOS
  3   Ø ' BLK CFA ' WORD !
  4   ' @ CFA ' WORD 2+ ! ;
  5
  6 ' FIXWORD CFA ' QUIT !
  7
  8 : LOAD DOS                   ( n -- )
  9   ' WORD D@ >R >R FIXWORD DROP
 1Ø   LOAD R> R> ' WORD D! ;
 11
 12 : (?LOADING)
 13   Ø= ' WORD @ ' BRANCH CFA <>
 14   AND ;  ' (?LOADING) CFA
 15   ' ?LOADING 4 + !               -->
```

```
Screen:  81
  Ø ( valDOS:  DIR                    )
  1
  2 : DIR DOS                      ( -- )
  3   DFLUNT 49 +  ALLNMS 2+ C!
  4   GETARGS Ø= IF ALLNMS ENDIF
  5   DUP 1+ C@ 248 AND 48 =
  6   IF
  7     1+ C@ 15 AND 48 + ALLNMS
  8     2+ C! ALLNMS
  9   ENDIF
 1Ø   FNCON ?SYSERR UNIT FSMGET
 11   CR CR ." Files on: "
 12   FSMAP 1Ø4 +
 13   BEGIN
 14     DUP C@ -DUP
 15   WHILE                          -->
```

```
Screen:  82
  Ø ( valDOS:  DIR                    )
  1     EMIT 1+
  2   REPEAT
  3   DROP CR CR
  4       ."    NAME     EXT    SIZE   S
  5 EC    ATTR  " CR
  6      ." +-+-+-+-+-+-+-+-+-+-+-+-+-+
  7 -+-+-+-+-+"
  8   CR CR CHKDIR DUP
  9   BEGIN
 1Ø     DUP Ø# ?TERMINAL Ø= AND
 11   WHILE
 12     2DUP - UNIT SWAP DIRTBL +
 13     C@ ENTRY SPACE DUP 5 + 8
 14     TYPE DUP 13 + C@ 32 <>
 15     IF 46 EMIT                  -->
```

```
Screen:  83
  Ø ( valDOS:  DIR                    )
  1
  2     ELSE SPACE ENDIF
  3     DUP 13 + 3 TYPE
  4     DUP 1+ @ 7 .R
  5     DUP 3 + @ 6 .R
  6     4 SPACES C@ DUP 32 AND
  7     IF 76 ( "L") EMIT ENDIF
  8     16 AND IF 82 ( "R") EMIT
  9     ENDIF 1- CR
 1Ø   REPEAT
 11   CR FSMAP 3 + @ .
 12   ." sectors free." CR CR
 13   DSKFLS 2DROP ;
 14
 15                               -->
```

```
Screen:  84
  Ø ( valDOS:  OPEN?                  )
  1
  2 : OPEN? DOS                  ( -- )
  3   CR Ø OUT ! MAXFL 1+ 1
  4   DO I FLBUF@ -DUP
  5     IF 8 + DUP C@
  6       SWAP 1+ C@
  7       I Ø <# # # #> TYPE
  8       OVER TO UNIT ENTRY
  9       5 + UNIT CONFN
 1Ø       2 SPACES $. CR
 11     ENDIF
 12   LOOP
 13   DIRFLS OUT @ Ø=
 14   IF ." No files open" CR ENDIF
 15   CR ;                        -->
```

```
Screen:  85
  Ø ( valDOS:  COPY                   )
  1
  2 : COPY DOS                   ( -- )
  3   GETARGS ?WRGARG
  4   61 GETARG ?WRGARG
  5   DUP (ENTER) Ø=
  6   IF
  7     DSKERR FLEXST = ?SYSERR
  8   ENDIF
  9   DUP (?OPEN) ?SYSERR
 1Ø   NOT FLOPN ?CMDERR
 11   (OPEN) ?SYSERR
 12   <ROT DROP OVER
 13   FLBUF@ C@ 32 AND
 14   Ø= FLWPRT ?CMDERR
 15                               -->
```

```
Screen:  86
  Ø ( valDOS:  COPY                   )
  1
  2   FNSWCH 65 ( "A") =
  3   IF
  4     Ø 3PICK (WIND) ?SYSERR
  5   ENDIF
  6   BEGIN
  7     Ø >R 44 ( ",") GETARG Ø=
  8     IF DUP R> 1+ >R ENDIF
  9     (OPEN) ?SYSERR SWAP DROP
 1Ø     BEGIN
 11       WRKSPC 128 3PICK (READ)
 12       DROP #UNTRN 128 <>
 13     WHILE
 14       WRKSPC 128 #UNTRN -
 15                               -->
```

```
Screen:  87
  Ø ( valDOS:  COPY  RENAME           )
  1
  2       5 PICK (WRITE) ?SYSERR
  3     REPEAT
  4     (CLOSE) R>
  5   UNTIL
  6   CR CR DROP DUP
  7   (ENDF) ?SYSERR
  8   (CLOSE) DSKFLS ;
  9
 1Ø : RENAME DOS                 ( -- )
 11   GETARGS ?WRGARG
 12   61 GETARG ?WRGARG
 13   SWAP (REN) ?SYSERR
 14   CR ." File renamed" CR CR
 15   DSKFLS ;                    -->
```

```
Screen:  88
  Ø ( valDOS:  system words          )
  1
  2 QUAN WLDFLG      QUAN SWITCH
  3 VECT DISKOP      VECT $OUT
  4 QUAN TEMP
  5
  6 : (CMDPAR) DOS                ( -- )
  7   GETARGS ?WRGARG
  8   BEGIN
  9     Ø >R 44 ( ",") GETARG Ø=
 10     IF DUP R> 1+ >R ENDIF
 11     DUP TO N1$ FNCON FNSWCH
 12     TO SWITCH ?WILD TO WLDFLG
 13     IF CHKDIR DUP DUP Ø=
 14       IF
 15         FNFLD UNIT CONFN       -->
```

```
Screen:  89
  Ø ( valDOS:  system words          )
  1
  2          CR Ø OUT ! $.
  3          18 OUT @ - SPACES
  4         ." is non-existent"
  5       ENDIF
  6       BEGIN -DUP
  7       WHILE
  8         DIRTBL OVER - 3PICK +
  9         C@ UNIT SWAP ENTRY 5 +
 10         UNIT CONFN 1 WLDFLG
 11         SWITCH 78 ( "N") <> *
 12         IF DROP CR $OUT SPACE
 13           DUP $. ."  ? " +Y/N
 14         ENDIF
 15           Ø OUT !              -->
```

```
Screen:  9Ø
  Ø ( valDOS:  system words          )
  1
  2         IF DUP DISKOP Ø= SWAP CR
  3           $. 18 OUT @ - SPACES
  4           IF
  5             DSKERR FLOPN =
  6             IF
  7               ." is open"
  8             ELSE
  9               DSKERR FLWPRT =
 10               IF
 11                 ." is locked"
 12               ELSE ." is ???"
 13               ENDIF
 14             ENDIF
 15           ELSE                  -->
```

```
Screen:  91
  Ø ( valDOS:  system words          )
  1
  2               ." is " $OUT ." ed"
  3             ENDIF
  4           ELSE
  5             DROP
  6           ENDIF 1-
  7         REPEAT
  8         DROP
  9       ELSE
 10         CR Ø OUT ! N1$ $.
 11         18 OUT @ - SPACES
 12         ." is illegal"
 13       ENDIF R>
 14     UNTIL
 15     DROP CR CR DSKFLS ;          -->
```

```
Screen:  92
  Ø ( valDOS:  KILL  LOCK            )
  1
  2 : $KILL ." kill" ;
  3
  4 : KILL DOS                    ( -- )
  5   ASSIGN (KILL) TO DISKOP
  6   ASSIGN $KILL TO $OUT
  7   (CMDPAR) ;
  8
  9
 10 : $LOCK ." lock" ;
 11
 12 : LOCK DOS                   ( -- )
 13   ASSIGN (LOCK) TO DISKOP
 14   ASSIGN $LOCK TO $OUT
 15   (CMDPAR) ;                     -->
```

```
Screen:  93
  Ø ( valDOS:  UNLOCK  SETUNIT       )
  1
  2 : $UNLOCK ." unlock" ;
  3
  4 : UNLOCK DOS                  ( -- )
  5   ASSIGN (UNLOCK) TO DISKOP
  6   ASSIGN $UNLOCK TO $OUT
  7   (CMDPAR) ;
  8
  9
 10 : SETUNIT DOS                 ( -- )
 11   GETARGS ?WRGARG GETVAL
 12   DUP 1 >= OVER 4 <= AND
 13   BADUNT ?CMDERR
 14   1- TO DFLUNT CR ;
 15                                  -->
```

```
Screen:  94
  Ø ( valDOS:  PRINT                    )
  1
  2 : PRINT DOS                    ( -- )
  3   GETARGS ?WRGARG 44 GETARG
  4   IF SWAP GETVAL 1- Ø MAX
  5   ELSE Ø ENDIF SWAP
  6   (OPEN) ?SYSERR SWAP DROP
  7   DUP <ROT OVER TO TEMP (SKIP)
  8   FNFLD UNIT CONFN
  9   CR CR ." File: " $. CR CR
 1Ø   FNSWCH 78 <> DUP TO FNSWCH
 11   TEMP 1+ * TO TEMP
 12   BEGIN
 13     Ø ?TERMINAL FNSWCH * Ø=
 14     IF DROP DUP (RDB) ENDIF
 15                                 -->
```

```
Screen:  95
  Ø ( valDOS:  PRINT                    )
  1
  2   WHILE
  3     FNSWCH 1 =
  4     IF Ø TO FNSWCH TEMP
  5       IF
  6         TEMP Ø
  7         <# # # # # #> TYPE
  8         SPACE 1 AT TEMP +!
  9       ENDIF
 1Ø     ENDIF
 11     DUP EMIT 155 = TO FNSWCH
 12   REPEAT
 13   (CLOSE)
 14   CR DSKFLS ;
 15                                 -->
```

```
Screen:  96
  Ø ( valDOS:  read/write utility   )
  1
  2 : FLOPEN DOS              ( $ -- )
  3   DUP 1+ C@ DUP 24Ø AND 48 =
  4   IF
  5     15 AND SWAP DROP Ø TO TEMP
  6   ELSE
  7     DROP TEMP 2 AND
  8     IF DUP (ENTER) Ø=
  9       IF
 1Ø         DSKERR FLEXST = ?SYSERR
 11       ENDIF
 12     ENDIF
 13     (OPEN) ?SYSERR SWAP DROP
 14   ENDIF ;
 15                                 -->
```

```
Screen:  97
  Ø ( valDOS:  READ                     )
  1
  2 : READ DOS                     ( -- )
  3   65535 GETARGS ?WRGARG
  4   44 GETARG ?WRGARG
  5   SWAP 44 GETARG
  6   IF
  7     SWAP 4 ROLL DROP
  8     GETVAL <ROT
  9   ENDIF
 1Ø   GETVAL <ROT
 11   1 TO TEMP FLOPEN (READ)
 12   Ø= DSKERR EOFERR <>
 13   AND NOT ?SYSERR
 14   TEMP IF FL# (CLOSE) ENDIF
 15   CR DSKFLS ;                       -->
```

```
Screen:  98
  Ø ( valDOS:  WRITE                    )
  1
  2 : WRITE DOS                    ( -- )
  3   GETARGS ?WRGARG
  4   44 GETARG ?WRGARG
  5   SWAP 44 GETARG ?WRGARG
  6   GETVAL SWAP GETVAL ROT
  7   2 TO TEMP FLOPEN (WRITE) Ø=
  8   IF
  9     DSKERR FLWPRT = ?SYSERR
 1Ø     FL# (CLOSE) FLWPRT CMDERR
 11   ENDIF
 12   TEMP IF
 13     FL# (ENDF) ?SYSERR
 14     FL# (CLOSE)
 15   ENDIF CR DSKFLS ;                 -->
```

```
Screen:  99
  Ø ( valDOS:  CLOSE                    )
  1
  2 : CLOSE DOS                    ( -- )
  3   Ø GETARGS
  4   IF
  5     SWAP DROP GETVAL
  6   ENDIF
  7   (CLOSE) CR DSKFLS ;
  8
  9
 1Ø
 11
 12
 13
 14
 15
```

Screen: 100
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 101
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 102
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 103
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 104
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Screen: 105
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

```
Screen: 106
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 107
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 108
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 109
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 110
 0 ( Utils:  OPEN  ENTER                    )
 1
 2 '( FLFL# )( 70 LOAD )
 3
 4 : OPEN DOS                          ( -- )
 5   GETARGS ?WRGARG
 6   (OPEN) ?SYSERR
 7   CR ." File access #" .
 8   DROP CR DSKFLS ;
 9
10 : ENTER DOS                         ( -- )
11   GETARGS ?WRGARG
12   (ENTER) ?SYSERR
13   CR DSKFLS ;
14
15                                        -->
```

```
Screen: 111
 0 ( Utils:  FSPACE  ENDFIL                 )
 1
 2 : FSPACE DOS                        ( -- )
 3   GETARGS ?WRGARG
 4   44 GETARG ?WRGARG
 5   GETVAL SWAP GETVAL
 6   SWAP (SPACE) ?SYSERR
 7   CR DSKFLS ;
 8
 9 : ENDFIL DOS                        ( -- )
10   GETARGS ?WRGARG
11   GETVAL (ENDF) ?SYSERR
12   CR DSKFLS ;
13
14
15                                        -->
```

```
Screen: 112
  Ø ( Utils:  FDUMP system words    )
  1
  2 : PSPACE                        ( -- )
  3   PFLAG @ DUP 254 AND
  4   PFLAG ! SPACE PFLAG ! ;
  5
  6 : P.                            ( c -- )
  7   127 AND DUP
  8   32 < OVER         ( ctrl? )
  9   124 > OR          ( clear...?)
 10   IF
 11     DROP 46 ( ".")
 12   ENDIF
 13   EMIT ;
 14
 15                                      -->
```

```
Screen: 113
  Ø ( Utils:  FDUMP                 )
  1
  2 : FDUMP DOS                     ( -- )
  3   GETARGS ?WRGARG
  4   (OPEN) ?SYSERR
  5   FNFLD UNIT CONFN
  6   CR CR ." File: " $. CR
  7   8 FNSWCH 87 = 1+ *
  8   TO FNSWCH SWAP DROP
  9   BASE @ HEX SWAP Ø
 10   BEGIN
 11     WRKSPC 128 4 PICK (READ)
 12     >R 128 #UNTRN - WRKSPC
 13     BEGIN
 14       OVER Ø> ?TERMINAL Ø= AND
 15                                     -->
```

```
Screen: 114
  Ø ( Utils:  FDUMP                 )
  1
  2     WHILE
  3       3 PICK FNSWCH MOD Ø=
  4       IF
  5         CR Ø TO TEMP
  6         Ø OUT ! 3 PICK Ø
  7         <# # # # # #>
  8         TYPE SPACE PSPACE
  9       ENDIF
 10       DUP C@ Ø <# # # #> TYPE
 11       SPACE 1+ ROT 1+
 12       ROT 1- ROT 1 AT TEMP +!
 13       OVER Ø= 4 PICK
 14       FNSWCH MOD Ø= OR
 15                                     -->
```

```
Screen: 115
  Ø ( Utils:  FDUMP                 )
  1
  2       IF
  3         FNSWCH 16 = 24 * 30 +
  4         OUT @ - SPACES PSPACE
  5         DUP TEMP - TEMP Ø
  6         DO DUP I + C@ P.
  7         LOOP DROP
  8       ENDIF
  9     REPEAT
 10     2DROP R> NOT
 11     ?TERMINAL OR
 12   UNTIL
 13   DROP (CLOSE)
 14   CR BASE ! DSKFLS ;
 15                                     -->
```

```
Screen: 116
  Ø ( Utils:  REWIND  EOF           )
  1
  2 : REWIND DOS                    ( -- )
  3   GETARGS ?WRGARG
  4   GETVAL 1 SWAP (WIND)
  5   ?SYSERR CR DSKFLS ;
  6
  7 : EOF DOS                       ( -- )
  8   GETARGS ?WRGARG
  9   GETVAL Ø SWAP (WIND)
 10   ?SYSERR CR DSKFLS ;
 11
 12
 13
 14
 15                                     -->
```

```
Screen: 117
  Ø ( Utils:  NAMEDISK              )
  1
  2 : NAMEDISK DOS                  ( -- )
  3   DFLUNT 1+ GETARGS
  4   IF SWAP DROP GETVAL ENDIF
  5   DUP 1 < OVER 4 > OR
  6   NOT BADUNT ?CMDERR
  7   1- FSMGET FSMAP 104 +
  8   CR ." Old: " DUP
  9   BEGIN DUP C@ -DUP
 10   WHILE EMIT 1+
 11   REPEAT
 12   DROP DUP 20 ERASE CR
 13   ." New: " 20 EXPECT CR CR
 14   1 TO ?FSMUP DSKFLS ;
 15
```

```
Screen: 118
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 119
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 120
 Ø ( Fmtrs:  system words                )
 1 '( FLFL# )( 7Ø LOAD )
 2
 3 : INSIN CR
 4   ." Insert source: <START> "
 5   BEGIN
 6     ?TERMINAL 1 =
 7   UNTIL
 8   46 EMIT ;
 9
10 : INSDST CR
11   ." Insert dest:  <SELECT> "
12   BEGIN
13     ?TERMINAL 2 =
14   UNTIL
15   46 EMIT ;                          -->
```

```
Screen: 121
 Ø ( Fmtrs:  FMOVE                       )
 1
 2 : FMOVE DOS                      ( -- )
 3   GETARGS ?WRGARG
 4   61 GETARG ?WRGARG
 5   Ø (CLOSE) SWAP INSIN
 6   (OPEN) ?SYSERR <ROT DROP
 7   DSKFLS INSDST DUP (ENTER) Ø=
 8   IF
 9     DSKERR FLEXST = ?SYSERR
10   ENDIF
11   (OPEN) ?SYSERR SWAP DROP
12   BEGIN
13     INSIN PAD DUP 1 AND +
14     TOPOM OVER - 4 PICK
15     (READ) INSDST PAD DUP            -->
```

```
Screen: 122
 Ø ( Fmtrs:  FMOVE                       )
 1
 2     1 AND + TADR OVER -
 3     4 PICK (WRITE) DSKFLS Ø=
 4     IF DROP (CLOSE) (CLOSE)
 5        DSKERR CMDERR
 6     ENDIF Ø=
 7   UNTIL
 8   DUP (ENDF) <ROT (CLOSE)
 9   (CLOSE) ?SYSERR CR CR DSKFLS ;
10
11 : GETNUM                        ( -- n )
12   HERE 1+ 1Ø EXPECT HERE 1+
13   BEGIN DUP C@ WHILE 1+ REPEAT
14   BL SWAP C! HERE NUMBER DROP ;
15                                       -->
```

```
Screen: 123
 Ø ( Fmtrs:  system words                )
 1
 2 : LOCKOUT DOS      ( u s cnt -- )
 3   ROT FSMGET 0+S
 4   DO
 5     I 1+ 8 /MOD FSMAP 1Ø + +
 6     SWAP 128
 7     BEGIN OVER
 8     WHILE 2/ SWAP 1- SWAP
 9     REPEAT
10     SWAP DROP DUP 255 XOR SWAP
11     3PICK C@ AND Ø# FSMAP 3 +
12     DUP @ ROT - SWAP !
13     OVER C@ AND SWAP C!
14   LOOP
15   1 TO ?FSMUP ;                      -->
```

```
Screen: 124
  0 ( Fmtrs:  FORMAT                    )
  1
  2 : FORMAT DOS                  ( -- )
  3   DFLUNT 1+ GETARGS
  4   IF GETVAL SWAP DROP ENDIF
  5   DUP DUP 1- TO UNIT CR CR
  6   ." Format unit " . ." ? " +Y/N
  7   IF
  8     DUP WRKSPC 772 ! (FMT)
  9     DROP WRKSPC 128 ERASE
 10     2 WRKSPC C!
 11     WRKSPC 1+ 707 OVER ! 716
 12     SWAP 2+ ! 15 WRKSPC 10 + C!
 13     WRKSPC 11 + 89 255 FILL
 14     1- 720* 359 +
 15     WRKSPC OVER 0 R/W          -->
```

```
Screen: 125
  0 ( Fmtrs:  FORMAT                    )
  1
  2     UNIT 359 9 LOCKOUT CR CR
  3     ." Diskname: "
  4     FSMAP 104 + 20 EXPECT CR CR
  5     ." Lock out error screens? "
  6     +Y/N
  7     IF
  8       UNIT 695 24 LOCKOUT
  9     ENDIF
 10     BEGIN CR CR
 11       ." Lock out sectors? "
 12       +Y/N DUP
 13       IF
 14         UNIT CR
 15         ." First sector: "     -->
```

```
Screen: 126
  0 ( Fmtrs:  FORMAT                    )
  1
  2         GETNUM CR
  3         ." # to lock out: "
  4         GETNUM LOCKOUT
  5       ENDIF
  6     0= UNTIL
  7   ENDIF
  8   DROP DSKFLS CR CR ;
  9
 10
 11
 12
 13
 14
 15                                  -->
```

```
Screen: 127
  0 ( Fmtrs:  system words              )
  1
  2 0 VARIABLE SEC/PAS
  3 0 VARIABLE SECNT
  4
  5 : AXLN  DOS                ( system )
  6   4 PICK 0
  7   DO 3PICK I 128 * +
  8      3PICK I + 3 PICK R/W
  9   LOOP 2DROP 2DROP ;
 10
 11 : DCSTP DOS                    ( -- )
 12   DSKFLS TOPOM PAD DUP 1 AND
 13   - - 0 128 U/ SWAP DROP
 14   SEC/PAS ! 0 SECNT ! ;
 15                                  -->
```

```
Screen: 128
  0 ( Fmtrs:  DISKCOPY1                 )
  1
  2 : DISKCOPY1                   ( -- )
  3   DCSTP
  4   BEGIN
  5     CR INSIN
  6     720 SECNT @ - SEC/PAS @ MIN
  7     DUP >R PAD DUP 1 AND - SECNT
  8     @ 2DUP 5 PICK <ROT 1 AXLN
  9     INSDST 0 AXLN CR
 10     R> SECNT +! SECNT @ DUP .
 11     ." sectors copied" 720 =
 12   UNTIL
 13   MTB CR ;
 14
 15                                  -->
```

```
Screen: 129
  0 ( Fmtrs:  DISKCOPY2                 )
  1
  2 : DISKCOPY2                   ( -- )
  3   DCSTP
  4   CR ." Insert source in drive 1
  5 " CR ." Insert dest.  in drive 2
  6 " CR ." Press START to copy"
  7   WAIT
  8   BEGIN
  9    720 SECNT @ - SEC/PAS @ MIN
 10    DUP >R PAD DUP 1 AND - SECNT
 11    @ 2DUP 5 PICK <ROT
 12    1 AXLN 720 + 0 AXLN
 13    R> SECNT +! SECNT @ 720 =
 14   UNTIL
 15   MTB CR ;
```

Screen: 130
```
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Screen: 131
```
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Screen: 132
```
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Screen: 133
```
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Screen: 134
```
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Screen: 135
```
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Screen: 136
```
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Screen: 137
```
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Screen: 138
```
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Screen: 139
```
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Screen: 140
```
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Screen: 141
```
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 142
  Ø ( +Y/N routine                          )
  1 ( This Y/N routine accepts upper
  2   or lower case and echos it.   )
  3
  4 BASE @ HEX
  5
  6 : +Y/N                          ( -- f )
  7   KEY DUP DF AND
  8   DUP 59 <>
  9   ØBRANCH [ ØØ14 , ]
 1Ø   DUP 4E <>
 11   ØBRANCH [ ØØØ8 , ]
 12   2DROP BRANCH [ FFDA , ]
 13   SWAP EMIT 59 = ;
 14
 15 BASE !
```

```
Screen: 143
  Ø
  1
  2
  3
  4
  5
  6
  7
  8
  9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen: 144
  Ø
  1
  2
  3
  4
  5
  6
  7
  8
  9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen: 145
  Ø
  1
  2
  3
  4
  5
  6
  7
  8
  9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen: 146
  Ø
  1
  2
  3
  4
  5
  6
  7
  8
  9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen: 147
  Ø
  1
  2
  3
  4
  5
  6
  7
  8
  9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen: 148
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 149
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 150
 0 ( Quan:  ASSIGN                    )
 1
 2
 3
 4 '( CFALIT
 5 : ASSIGN [COMPILE] CFALIT ;
 6 IMMEDIATE --> )(  )
 7
 8 : ASSIGN                  ( -- cfa )
 9   STATE @
10   [COMPILE] [
11   [COMPILE] ' CFA SWAP
12   IF ]
13   ENDIF [COMPILE] LITERAL ;
14   IMMEDIATE
15                                  -->
```

```
Screen: 151
 0 ( Quan:  TO  AT                    )
 1
 2 : TO
 3   -FIND 0= 0 ?ERROR DROP
 4   STATE @
 5   IF ,
 6   ELSE EXECUTE
 7   ENDIF ; IMMEDIATE
 8
 9 : AT
10   -FIND 0= 0 ?ERROR DROP
11   2+ STATE @
12   IF ,
13   ELSE EXECUTE
14   ENDIF ; IMMEDIATE
15   ( corrected )                  -->
```

```
Screen: 152
 0 ( Quan:  [2@6]  [2!4]              )
 1
 2 ASSEMBLER HEX
 3
 4 LABEL (2@6)
 5   A0 C, 06 C, B1 C, W  C, 48 C,
 6   C8 C, B1 C, W  C, 4C C, PUSH ,
 7
 8 LABEL (2!4)
 9   A0 C, 04 C, B5 C, 00 C, 91 C,
10   W  C, C8 C, B5 C, 01 C, 91 C,
11   W  C, 4C C, POP ,
12
13
14
15                                  -->
```

```
Screen: 153
 0 ( Quan:  [2V6]                     )
 1
 2 LABEL (2V6)
 3   A0 C, 07 C, B1 C, W  C, 48 C,
 4   88 C, B1 C, W  C, 85 C, W  C,
 5   68 C, 85 C, W 1+ C,
 6   A0 C, 00 C, 4C C, W 1- ,
 7
 8
 9
10
11
12
13
14
15                                  -->
```

```
Screen: 154
  0 ( Quan:  patch for CREATE       )
  1
  2 DCX
  3
  4 : (PTCH)                   ( system )
  5   SWAP >R R = 251 R = 249 R> =
  6   OR OR ;
  7
  8 : PTCH                     ( system )
  9   IF [ ' (PTCH) CFA ] LITERAL
 10   ELSE [ ' = CFA ] LITERAL
 11   ENDIF
 12   [ ' CREATE 63 + ] LITERAL ! ;
 13
 14
 15                                   -->
```

```
Screen: 155
  0 ( Quan:  QUAN  VECT             )
  1
  2 : QUAN
  3   ON PTCH LABEL -2 ALLOT
  4   (2@6) , (2!4) ,
  5   [ ' VARIABLE 4 + ] LITERAL ,
  6   2 ALLOT OFF PTCH ;
  7
  8 : VECT
  9   ON PTCH LABEL -2 ALLOT
 10   (2V6) , (2!4) ,
 11   [ ' VARIABLE 4 + ] LITERAL ,
 12   [ ' NOOP CFA ] LITERAL ,
 13   OFF PTCH ;
 14
 15
```

```
Screen: 156
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 157
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 158
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 159
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 160
 0 ( Utils:  CARRAY  ARRAY          )
 1 BASE @ HEX
 2 : CARRAY          ( cccc, n -- )
 3   CREATE SMUDGE ( cccc: n -- a )
 4     ALLOT
 5   ;CODE CA C, CA C, 18 C,
 6   A5 C, W C,  69 C, 02 C, 95 C,
 7   00 C, 98 C, 65 C, W 1+ C,
 8   95 C, 01 C, 4C C,
 9   ' + ( CFA @ ) , C;
10
11 : ARRAY          ( cccc, n -- )
12   CREATE SMUDGE ( cccc: n -- a )
13     2* ALLOT
14   ;CODE 16 C, 00 C, 36 C, 01 C,
15    4C C, ' CARRAY 08 + , C;  -->
```

```
Screen: 161
 0 ( Utils:  CTABLE  TABLE          )
 1
 2 : CTABLE          ( cccc, -- )
 3   CREATE SMUDGE ( cccc: n -- a )
 4   ;CODE
 5   4C C, ' CARRAY 08 + , C;
 6
 7 : TABLE           ( cccc, -- )
 8   CREATE SMUDGE ( cccc: n -- a )
 9   ;CODE
10   4C C, ' ARRAY 0A + , C;
11
12
13
14
15                                -->
```

```
Screen: 162
 0 ( Utils:  2CARRAY  2ARRAY        )
 1
 2 : 2CARRAY       ( cccc, n n -- )
 3   <BUILDS       ( cccc: n n -- a )
 4    SWAP DUP , * ALLOT
 5   DOES>
 6     DUP >R @ * + R> + 2+ ;
 7
 8 : 2ARRAY        ( cccc, n n -- )
 9   <BUILDS       ( cccc: n n -- a )
10    SWAP DUP , * 2* ALLOT
11   DOES>
12     DUP >R @ * + 2* R> + 2+ ;
13
14
15                                -->
```

```
Screen: 163
 0 ( Utils:  XC!  X!                )
 1
 2 : XC!     ( n0...nm cnt addr -- )
 3   OVER 1- + >R 0
 4   DO J I - C!
 5   LOOP R> DROP ;
 6
 7 : X!      ( n0...nm cnt addr -- )
 8   OVER 1- 2* + >R 0
 9   DO J I 2* - !
10   LOOP R> DROP ;
11
12 ( Caution: Remember limitation
13 ( on stack size of 30 values
14 ( because of OS conflict. )
15                                -->
```

```
Screen: 164
 0 ( Utils:  CVECTOR  VECTOR        )
 1
 2 : CVECTOR         ( cccc, cnt -- )
 3   CREATE SMUDGE ( cccc: n -- a )
 4   HERE OVER ALLOT XC!
 5   ;CODE
 6   4C C, ' CARRAY 08 + , C;
 7
 8 : VECTOR          ( cccc, cnt -- )
 9   CREATE SMUDGE ( cccc: n -- a )
10   HERE OVER 2* ALLOT X!
11   ;CODE
12   4C C, ' ARRAY  0A + , C;
13
14 BASE !
15
```

```
Screen: 165
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Screen: 166
```
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Screen: 167
```
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Screen: 168
```
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Screen: 169
```
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Screen: 170
```
 Ø CONTENTS OF THIS DISK:
 1
 2 Disk Operating System:    10 LOAD
 3 DOS system extensions:    50 LOAD
 4 BASIC DOS COMMANDS:       70 LOAD
 5 DOS COMMAND EXTENSIONS:  110 LOAD
 6
 7 (note the packages lower on the
 8  screen will load all packages
 9  listed above themselves.)
10
11 DISK FORMATTER/COPIERS:  120 LOAD
12 QUAN STRUCTURES:         150 LOAD
13 ARRAYS & THEIR COUSINS:  160 LOAD
14
15 valDOS FILE EDITOR:     valDOS II
```

Screen: 171
```
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 172
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen: 173
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen: 174
 Ø ( valDOS error messages                )
 1 Illegal filename
 2 Bad/Mismatched unit(s)
 3 Bad free space map
 4 File already exists
 5 Directory is full
 6 Disk is full
 7 Filename is ambiguous
 8 File does not exist
 9 No room for buffer
 1Ø End of file encountered
 11 File is not open
 12 Illegal file number
 13 File is locked
 14 Bad argument list
 15 File is open
```

```
Screen: 175
 Ø File is too big
 1 File is random
 2 File is not random
 3 No room for random map
 4 Random map is bad
 5 File is a device file
 6 File is not a device file
 7 Illegal access to device file
 8
 9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen: 176
 Ø ( Error messages                       )
 1
 2 Stack empty
 3
 4 Dictionary full
 5
 6 Wrong addressing mode
 7
 8 Is not unique
 9
 1Ø Value error
 11
 12 Disk address error
 13
 14 Stack full
 15
```

```
Screen: 177
 Ø Disk Error!
 1
 2 Dictionary too big
 3
 4
 5
 6
 7
 8
 9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen: 178
  Ø ( Error messages                    )
  1
  2 Use only in Definitions
  3
  4 Execution only
  5
  6 Conditionals not paired
  7
  8 Definition not finished
  9
 1Ø In protected dictionary
 11
 12 Use only when loading
 13
 14 Off current screen
 15


Screen: 179
  Ø Declare VOCABULARY
  1
  2
  3
  4
  5
  6
  7
  8
  9
 1Ø
 11
 12
 13
 14
 15
```

valDOS II

Supplied Source Listing

LXVI.

```
Screen:  50
  0 ( Case Statements:  CASE          )
  1 BASE @ DCX
  2
  3 '( PERMANENT PERMANENT )(   )
  4 : (CASE)
  5   R C@ MIN -1 MAX 2*
  6   R 3 + + @EX
  7   R C@ 2* 5 + R> + >R ;
  8
  9 '( TRANSIENT TRANSIENT )(   )
 10 : CASE
 11   ?COMP COMPILE (CASE)
 12   HERE 0 C,
 13   COMPILE NOOP 6 ;     IMMEDIATE
 14
 15 '( PERMANENT PERMANENT )(   ) -->
```

```
Screen:  51
  0 ( Case statements:  CASE          )
  1
  2 : NOCASE
  3   6 ?PAIRS 7 ;          IMMEDIATE
  4
  5 : CASEND
  6   DUP 6 =
  7   IF DROP COMPILE NOOP
  8   ELSE 7 ?PAIRS
  9   ENDIF
 10   HERE 2- @ OVER 1+ !
 11   HERE OVER -
 12   5 - 2/ SWAP C! ;      IMMEDIATE
 13
 14 '( PERMANENT PERMANENT )(   )
 15                                 -->
```

```
Screen:  52
  0 ( Case statements:  SEL           )
  1
  2 : (SEL)
  3   R 1+ DUP 2+ DUP R C@
  4   2* 2* + R> DROP DUP >R SWAP
  5   DO I @ 3 PICK =
  6     IF I 2+ SWAP DROP LEAVE THEN
  7   4 /LOOP SWAP DROP @EX ;
  8
  9 '( TRANSIENT TRANSIENT )(   )
 10 : SEL     ?COMP
 11   ?LOADING COMPILE (SEL) HERE
 12   0 C, COMPILE NOOP [COMPILE] [
 13   8 ; IMMEDIATE
 14
 15 '( PERMANENT PERMANENT )(   ) -->
```

```
Screen:  53
  0 ( Case statements:  SEL           )
  1
  2 : NOSEL
  3   8 ?PAIRS [COMPILE] ' CFA
  4   OVER 1+ ! 8 ;          IMMEDIATE
  5
  6 : ->
  7   SWAP 8 ?PAIRS , DUP C@ 1+
  8   OVER C! [COMPILE] '
  9   CFA , 8 ;              IMMEDIATE
 10
 11 : SELEND
 12   8 ?PAIRS
 13   DROP [COMPILE] ] ;     IMMEDIATE
 14
 15                                 -->
```

```
Screen:  54
  0 ( Case statements:  COND          )
  1
  2 '( TRANSIENT TRANSIENT )(   )
  3
  4 : COND
  5   0 COMPILE DUP ;        IMMEDIATE
  6
  7 : <<
  8   1+ [COMPILE] IF
  9   COMPILE DROP ;         IMMEDIATE
 10
 11 : >>
 12   [COMPILE] ELSE COMPILE
 13   DUP ROT ;              IMMEDIATE
 14
 15 '( PERMANENT PERMANENT )(   ) -->
```

```
Screen:  55
  0 ( Case statements:  COND          )
  1
  2 : NOCOND
  3   COMPILE 2DROP ;        IMMEDIATE
  4
  5 : CONDEND
  6   0 DO
  7     [COMPILE] ENDIF
  8   LOOP ;                 IMMEDIATE
  9
 10
 11
 12
 13
 14
 15                                 -->
```

```
Screen:  56
  0 ( Case statements:  CASE:       )
  1
  2 '( PERMANENT PERMANENT )(   )
  3
  4 : CASE:
  5   <BUILDS
  6     SMUDGE !CSP
  7     [COMPILE] ]
  8   DOES>
  9     SWAP 2* + @EX ;
 10
 11
 12 BASE !
 13
 14
 15
```

```
Screen:  57
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen:  58
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen:  59
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen:  60
  0 ( Screen code conversion words )
  1
  2 BASE @ HEX
  3
  4 CODE >BSCD            ( a a n -- )
  5   A9 C, 03 C, 20 C, SETUP ,
  6   HERE  C4 C, C2 C, D0 C, 07 C,
  7   C6 C, C3 C, 10 C, 03 C, 4C C,
  8   NEXT ,        B1 C, C6 C, 48 C,
  9   29 C, 7F C, C9 C, 60 C, B0 C,
 10   0D C, C9 C, 20 C, B0 C, 06 C,
 11   18 C, 69 C, 40 C, 4C C, HERE
 12 2 ALLOT 38 C, E9 C, 20 C, HERE
 13 SWAP !  91 C, C4 C, 68 C, 29 C,
 14   80 C, 11 C, C4 C, 91 C, C4 C,
 15                              -->
```

```
Screen:  61
  0 ( Screen code conversion words )
  1
  2   C8 C, D0 C, D3 C, E6 C, C7 C,
  3   E6 C, C5 C, 4C C, ,
  4 C;
  5
  6 CODE BSCD>            ( a a n -- )
  7   A9 C, 03 C, 20 C, SETUP ,
  8   HERE  C4 C, C2 C, D0 C, 07 C,
  9   C6 C, C3 C, 10 C, 03 C, 4C C,
 10   NEXT ,        B1 C, C6 C, 48 C,
 11   29 C, 7F C, C9 C, 60 C, B0 C,
 12   0D C, C9 C, 40 C, B0 C, 06 C,
 13   18 C, 69 C, 20 C, 4C C, HERE
 14 2 ALLOT 38 C, E9 C, 40 C, HERE
 15                              -->
```

```
Screen:  62
  Ø ( Screen code conversion words )
  1
  2 SWAP !  91 C, C4 C, 68 C, 29 C,
  3   8Ø C, 11 C, C4 C, 91 C, C4 C,
  4   C8 C, DØ C, D3 C, E6 C, C7 C,
  5   E6 C, C5 C, 4C C, ,
  6 C;
  7
  8 : >SCD                  ( c -- sc )
  9   SP@ DUP 1 >BSCD ;
 1Ø
 11 : SCD>                  ( sc -- c )
 12   SP@ DUP 1 BSCD> ;
 13
 14
 15 BASE !
```

```
Screen:  63
  Ø
  1
  2
  3
  4
  5
  6
  7
  8
  9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen:  64
  Ø
  1
  2
  3
  4
  5
  6
  7
  8
  9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen:  65
  Ø
  1
  2
  3
  4
  5
  6
  7
  8
  9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen:  66
  Ø
  1
  2
  3
  4
  5
  6
  7
  8
  9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen:  67
  Ø
  1
  2
  3
  4
  5
  6
  7
  8
  9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen:  68
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen:  69
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen:  70
 0 ( Utils:  HIDCHR  NOKEY  CURSOR)
 1
 2 BASE @ DCX
 3
 4 '( CASE )( 50 LOAD )
 5
 6 : HIDCHR                    ( -- )
 7   65535 94 ! ;
 8
 9 : NOKEY                     ( -- )
10   255 764 C! ;
11
12 : CURSOR                  ( f -- )
13   0= 752 C!
14   28 EMIT 29 EMIT ;
15                                -->
```

```
Screen:  71
 0 ( Utils:  INKEY$                 )
 1
 2 : (INKEY$)                  ( c -- )
 3   702 C! NOKEY @ ;
 4
 5 : INKEY$                    ( -- c )
 6   764 C@
 7   COND
 8     252 = << 128 (INKEY$) >>
 9     191 > <<   0 >>
10     188 = <<   0 >>
11     124 = << 64 (INKEY$) >>
12      60 = <<   0 (INKEY$) >>
13      39 = <<   0 NOKEY    >>
14   NOCOND KEY
15   CONDEND ;                        -->
```

```
Screen:  72
 0 ( Utils:  -Y/N                   )
 1
 2 ( This Y/N routine accepts upper
 3   or lower case.  )
 4 HEX
 5
 6 : -Y/N                      ( -- f )
 7   KEY DF AND
 8   DUP 59 <>
 9   0BRANCH [ 0014 , ]
10   DUP 4E <>
11   0BRANCH [ 0008 , ]
12   2DROP BRANCH [ FFDA , ]
13   59 = ;
14
15 DCX                               -->
```

```
Screen:  73
 0 ( Utils:  Y/N  -RETURN  RETURN )
 1
 2 : Y/N                       ( -- f )
 3   ." <Y/N> " -Y/N DUP
 4   IF 89 ELSE 78 ENDIF
 5   EMIT SPACE ;
 6
 7 : -RETURN                     ( -- )
 8   BEGIN
 9     KEY 155 =
10   UNTIL ;
11
12 : RETURN                     ( -- )
13   ." <RETURN> " -RETURN ;
14
15 BASE !
```

```
Screen: 100
  0 ( valDOS file editor 1.0          )
  1 '( FLFL# )( 160 LOAD ;S )
  2
  3 BASE @ DCX
  4
  5 VOCABULARY EDITOR IMMEDIATE
  6 EDITOR DEFINITIONS
  7
  8 '( >BSCD  )( 60 LOAD )
  9 '( HIDCHR )( 70 LOAD )
 10
 11 QUAN XLOC       0 TO XLOC
 12 QUAN YLOC       0 TO YLOC
 13 QUAN INSRT      0 TO INSRT
 14 QUAN LSTCHR     0 TO LSTCHR
 15 QUAN ?BUFSM     0 TO ?BUFSM    -->
```

```
Screen: 101
  0 ( valDOS file editor 1.0          )
  1
  2 QUAN ?PADSM      0 TO ?PADSM
  3 QUAN ?MULTI      0 TO ?MULTI
  4 QUAN ?MARK       0 TO ?MARK
  5 QUAN ?UPDAT      0 TO ?UPDAT
  6 QUAN TXTBOT      0 TO TXTBOT
  7 QUAN TXTEND      0 TO TXTEND
  8 QUAN DSPTOP      0 TO DSPTOP
  9 QUAN DSPBOT      0 TO DSPBOT
 10 QUAN LNCNT       0 TO LNCNT
 11 QUAN CURLN       0 TO CURLN
 12 QUAN EDFL#       0 TO EDFL#
 13 QUAN MEMTOP      0 TO MEMTOP
 14 QUAN PTFLG       0 TO PTFLG
 15 QUAN ?2BIG       0 TO ?2BIG     -->
```

```
Screen: 102
  0 ( valDOS file editor 1.0          )
  1
  2 LABEL OOPSLN 38 ALLOT
  3 OOPSLN 38 ERASE
  4 LABEL EDN1$ 16 ALLOT
  5 LABEL EDN2$ 16 ALLOT
  6 LABEL SRCH$ 34 ALLOT 0 SRCH$ C!
  7 LABEL EDWRK 40 ALLOT
  8 LABEL TABS 4 ALLOT 32 C,
  9 TABS 4 68 FILL
 10 36 TABS 4 + C!
 11
 12 37 CONSTANT 37
 13 16 CONSTANT #LNS
 14 40 CONSTANT LLEN
 15 5 LLEN * CONSTANT BLEN          -->
```

```
Screen: 103
  0 ( valDOS file editor 1.0          )
  1 HEX
  2
  3 LABEL TOS1+
  4   F6 C, 00 C, D0 C, 02 C, F6 C,
  5   01 C, 60 C,
  6
  7 LABEL TOS1-
  8   B5 C, 00 C, D0 C, 02 C, D6 C,
  9   01 C, D6 C, 00 C, 60 C,
 10
 11 CODE PRVLN                ( a -- a )
 12  20 C, TOS1- , A1 C, 00 C, F0 C,
 13  09 C, 20 C, TOS1- , A1 C, 00 C,
 14  C9 C, 9B C,   D0 C, F3 C, 20 C,
 15  TOS1+ , 4C C, NEXT , C;        -->
```

```
Screen: 104
  0 ( valDOS file editor 1.0          )
  1
  2 CODE NXTLN                 ( a -- a )
  3  A1 C,   00 C, F0 C, 07 C, 20 C,
  4  TOS1+ , C9 C, 9B C, D0 C, F5 C,
  5  4C C, NEXT , C;
  6
  7 CODE CRAM                ( A A -- L )
  8   A9 C, 02 C, 20 C, SETUP ,
  9   A0 C, 26 C, 88 C, F0 C, 04 C,
 10   B1 C, N 2+ C, F0 C, -7 C,
 11   C8 C, 84 C, XSAVE C, A9 C,
 12   DB C, 91 C, N C, 88 C, 30 C,
 13   07 C, B1 C, N 2+ C, 91 C,
 14   N  C, 4C C, HERE 8 - , A4 C,
 15                                 -->
```

```
Screen: 105
  0 ( valDOS file editor 1.0          )
  1
  2   XSAVE C, C8 C, 98 C,
  3   4C C, PUSH0A ,
  4 C;
  5
  6 CODE UNCRAM              ( a a -- a )
  7   A9 C, 02 C, 20 C, SETUP ,
  8   B1 C, N 2+ C, F0 C, 0D C,
  9   B1 C, N 2+ C, C9 C, 9B C,
 10   F0 C, 06 C, 91 C, N C,
 11   C8 C, 4C C, HERE 0A - ,
 12   C8 C, 18 C, 98 C, 65 C,
 13   N 2+ C, 48 C, A9 C, 00 C,
 14   65 C, N 3 + C, 4C C, PUSH ,
 15 C;                              -->
```

```
Screen: 106
 0 ( valDOS file editor 1.0          )
 1
 2 CODE ?ENUF          ( a -- a cnt )
 3  84 C, N  C, A1 C, 00 C, C9 C,
 4  9B C, D0 C, 0A C, E6 C, N  C,
 5  A9 C, 10 C, C5 C, N  C, F0 C,
 6  0E C, D0 C, 03 C, 0A C, F0 C,
 7  09 C, F6 C, 00 C, D0 C, 02 C,
 8  F6 C, 01 C, 4C C, ' ?ENUF 2+ ,
 9  A5 C, N  C, 4C C, PUSH0A ,
10 C;
11
12 DCX
13
14
15                                -->
```

```
Screen: 107
 0 ( valDOS file editor 1.0          )
 1
 2 : LMOVE                  ( f t -- )
 3   40 CMOVE ;
 4
 5 : BOL                      ( -- a )
 6   88 @ YLOC 1+ LLEN * + 2+ ;
 7
 8 : SBL                      ( -- a )
 9   88 @ LLEN #LNS 1+ * + ;
10
11 : PBL                      ( -- a )
12   PAD 128 + ;
13
14 : PBLL                     ( -- a )
15   PBL BLEN + LLEN - ;          -->
```

```
Screen: 108
 0 ( valDOS file editor 1.0          )
 1
 2 : TOPLN                    ( -- a )
 3   88 @ LLEN + 2+ ;
 4
 5 : BOTLN                    ( -- a )
 6   88 @ #LNS LLEN * + 2+ ;
 7
 8 : CURLOC                   ( -- a )
 9   BOL XLOC + ;
10
11 : CSHOW                    ( -- a )
12   CURLOC DUP C@
13   128 OR SWAP C! ;
14
15                                -->
```

```
Screen: 109
 0 ( valDOS file editor 1.0          )
 1
 2 : CBLANK                    ( -- )
 3   CURLOC DUP C@
 4   127 AND SWAP C! ;
 5
 6 : NORPT 0 TO ?MULTI ;       ( -- )
 7
 8 : CRWT                      ( -- )
 9   CR ." C/R to continue"
10   253 EMIT
11   BEGIN
12     KEY 155 =
13   UNTIL
14   88 @ 40 + 640 ERASE ;
15                                -->
```

```
Screen: 110
 0 ( valDOS file editor 1.0          )
 1
 2 : OOPSV                     ( -- )
 3   BOL OOPSLN 38 CMOVE ;
 4
 5
 6 : ?FULL                     ( -- )
 7   MEMTOP 1- TXTEND U<
 8   DUP TO ?2BIG
 9   IF
10     1500 0
11     DO
12       0 53279 C!
13     LOOP
14   ENDIF ;
15                                -->
```

```
Screen: 111
 0 ( valDOS file editor 1.0          )
 1
 2 : SAVLN                     ( -- )
 3   ?MARK ?2BIG NOT AND
 4   IF 0 TO ?MARK BOL PAD CRAM
 5     DUP LNCNT <>
 6     IF CURLN NXTLN DUP 3 PICK
 7       LNCNT - DUP >R +
 8       TXTEND 3 PICK - R>
 9       DUP AT TXTEND +! 0<
10       IF CMOVE
11       ELSE <CMOVE ENDIF ?FULL
12     ENDIF
13     PAD CURLN ROT BSCD>
14   ENDIF ;
15                                -->
```

```
Screen: 112
  Ø ( valDOS file editor 1.Ø          )
  1
  2 : DISPLAY                      ( -- )
  3   TOPLN 2- DSPTOP
  4   16 Ø
  5   DO
  6     PAD 4Ø BLANKS DUP
  7     PAD 2+ UNCRAM DUP ROT =
  8     IF
  9       OVER 4Ø 64 FILL
 1Ø     ELSE
 11       PAD 3 PICK LLEN >BSCD
 12     ENDIF
 13     SWAP 4Ø + SWAP
 14   LOOP
 15   2DROP 1 752 C! ;              -->
```

```
Screen: 113
  Ø ( valDOS file editor 1.Ø          )
  1
  2 : GETLN                        ( -- )
  3   CURLN DUP NXTLN
  4   SWAP - TO LNCNT ;
  5
  6 : GC     GETLN CSHOW ;         ( -- )
  7
  8 : DC     DISPLAY CSHOW ;       ( -- )
  9
 1Ø : CS     CBLANK SAVLN ;        ( -- )
 11
 12 : HOME                    ( dtop -- )
 13   DUP TO DSPTOP TO CURLN
 14   Ø TO XLOC Ø TO YLOC
 15   DISPLAY GC ;                  -->
```

```
Screen: 114
  Ø ( valDOS file editor 1.Ø          )
  1
  2 : ROLLDN                       ( -- )
  3   DSPTOP DUP
  4   ?ENUF 2DROP 2- Ø
  5   IF
  6     CS DSPTOP PRVLN
  7     DUP TO DSPTOP
  8     PAD 4Ø BLANKS PAD 2+
  9     UNCRAM DROP PAD
 1Ø     TOPLN 2- DUP DUP 4Ø +
 11     6ØØ <CMOVE LLEN >BSCD
 12     CURLN PRVLN TO CURLN GC
 13   ELSE
 14     NORPT
 15   ENDIF ;                       -->
```

```
Screen: 115
  Ø ( valDOS file editor 1.Ø          )
  1
  2 : ROLLUP                       ( -- )
  3   DSPTOP CS
  4   ?ENUF SWAP 1+ TO DSPBOT
  5   16 <> DSPBOT C@ Ø= OR NOT
  6   IF
  7     CURLN NXTLN TO CURLN
  8     TOPLN 2- DUP 4Ø + SWAP 6ØØ
  9     CMOVE PAD 4Ø BLANKS
 1Ø     DSPBOT PAD 2+ UNCRAM DROP
 11     PAD BOTLN 2- LLEN >BSCD
 12     DSPTOP NXTLN TO DSPTOP
 13   ELSE
 14     NORPT
 15   ENDIF GC ;                    -->
```

```
Screen: 116
  Ø ( valDOS file editor 1.Ø          )
  1
  2 : UPCUR                        ( -- )
  3   ?MULTI Ø= YLOC OR
  4   IF YLOC
  5     IF
  6       CS CURLN PRVLN TO CURLN
  7       -1 AT YLOC +! GC
  8     ELSE
  9       ROLLDN
 1Ø     ENDIF
 11   ELSE
 12     NORPT
 13   ENDIF ;
 14
 15                                 -->
```

```
Screen: 117
  Ø ( valDOS file editor 1.Ø          )
  1
  2 : DNCUR                        ( -- )
  3   ?MULTI Ø= YLOC 15 <> OR
  4   IF YLOC #LNS 1- =
  5     IF
  6       ROLLUP
  7     ELSE
  8       CURLOC 4Ø + C@ 64 <>
  9       IF
 1Ø         CS CURLN NXTLN TO CURLN
 11         1 AT YLOC +! GETLN
 12       ENDIF
 13     ENDIF
 14   ELSE NORPT
 15   ENDIF CSHOW ;                 -->
```

```
Screen: 118
  0 ( valDOS file editor 1.0          )
  1
  2 : LFCUR                          ( -- )
  3   XLOC 1- DUP 0<
  4   IF UPCUR DROP 37 ENDIF
  5   CBLANK TO XLOC CSHOW ?MULTI
  6   IF XLOC TO ?MULTI ENDIF ;
  7
  8 : RTCUR                          ( -- )
  9   XLOC 1+ DUP 37 >
 10   IF DROP 0 DNCUR ENDIF
 11   CBLANK TO XLOC CSHOW ?MULTI
 12   IF XLOC 37 <> TO ?MULTI THEN ;
 13
 14 : EDMRK
 15   1 TO ?MARK 1 TO ?UPDAT ;     -->
```

```
Screen: 119
  0 ( valDOS file editor 1.0          )
  1
  2 : INTGL                          ( -- )
  3   INSRT NOT TO INSRT NORPT ;
  4
  5 : CLREOL                         ( -- )
  6   CBLANK OOPSV CURLOC 38 XLOC -
  7   ERASE CSHOW EDMRK NORPT ;
  8
  9 HEX
 10 CODE 2^                  ( n -- 2^n )
 11   B4 C, 00 C, C8 C, A9 C, 00 C,
 12   95 C, 00 C, 95 C, 01 C, 38 C,
 13   36 C, 00 C, 36 C, 01 C, 18 C,
 14   88 C, D0 C, F8 C, 4C C, NEXT ,
 15  C; DCX                          -->
```

```
Screen: 120
  0 ( valDOS file editor 1.0          )
  1
  2 : BYTINS                         ( -- )
  3   CBLANK XLOC 37 <
  4   IF
  5     CURLOC DUP 1+
  6     37 XLOC - <CMOVE
  7   ENDIF
  8   0 CURLOC C!
  9   CSHOW EDMRK ;
 10
 11
 12
 13
 14
 15                                  -->
```

```
Screen: 121
  0 ( valDOS file editor 1.0          )
  1
  2 : BYTDEL                         ( -- )
  3   CBLANK
  4   XLOC 37 <
  5   IF
  6     CURLOC DUP 1+ SWAP
  7     37 XLOC - CMOVE
  8   ENDIF
  9   0 CURLOC
 10   37 XLOC - + C!
 11   CSHOW EDMRK ;
 12
 13
 14
 15                                  -->
```

```
Screen: 122
  0 ( valDOS file editor 1.0          )
  1
  2 : LNINS                          ( -- )
  3   ?2BIG NOT
  4   IF
  5     CS YLOC 15 <
  6     IF BOL 2- DUP LLEN + #LNS 1-
  7        YLOC - LLEN * <CMOVE
  8     ENDIF
  9     BOL 2- 40 ERASE CURLN
 10     DUP TXTEND OVER - 1+
 11     OVER 1+ SWAP <CMOVE 155
 12     SWAP C! 1 AT TXTEND +!
 13     GC 1 TO ?UPDAT
 14   ENDIF
 15   ?FULL ;                        -->
```

```
Screen: 123
  0 ( valDOS file editor 1.0          )
  1
  2 : BONXT                          ( -- )
  3   CS 0 TO XLOC
  4   CURLN NXTLN C@ 0= PTFLG OR
  5   ?2BIG NOT AND
  6   IF
  7     CURLN NXTLN
  8     DUP DUP 1+ TXTEND 3 PICK
  9     - 2+ <CMOVE 155 SWAP C!
 10     1 AT TXTEND +!
 11     DISPLAY ?FULL
 12     1 TO ?UPDAT
 13   ENDIF
 14   DNCUR
 15   0 TO PTFLG ;                   -->
```

```
Screen: 124
  Ø ( valDOS file editor 1.Ø          )
  1
  2 : LNDEL                          ( -- )
  3   YLOC CURLOC 4Ø + C@
  4   64 <> OR
  5   IF CBLANK OOPSV
  6     CURLN DUP NXTLN 2DUP
  7     SWAP TXTEND 3 PICK - 2+
  8     CMOVE - AT TXTEND +!
  9     Ø TO ?MARK GETLN DISPLAY
 1Ø     CURLOC C@ 64 =
 11     IF UPCUR NORPT ENDIF
 12     CSHOW 1 TO ?UPDAT
 13   ELSE
 14     NORPT
 15   ENDIF ;                            -->
```

```
Screen: 125
  Ø ( valDOS file editor 1.Ø          )
  1
  2 : BFSHW                          ( -- )
  3   PBLL LLEN 2* 2* -
  4   SBL LLEN 5 * CMOVE ;
  5
  6 : BFROT                          ( -- )
  7   PBL DUP BLEN + LMOVE
  8   PBL DUP LLEN + SWAP
  9   BLEN LLEN - CMOVE
 1Ø   PBLL LLEN + PBLL LMOVE
 11   BFSHW ;
 12
 13 : BFCPY                          ( -- )
 14   CBLANK BFROT BOL PBLL 2+
 15   38 CMOVE BFSHW CSHOW ;             -->
```

```
Screen: 126
  Ø ( valDOS file editor 1.Ø          )
  1
  2 : <BFROT                         ( -- )
  3   PBLL DUP LLEN + LMOVE
  4   PBL DUP LLEN +
  5   BLEN LLEN - <CMOVE
  6   PBL DUP BLEN +
  7   SWAP LMOVE BFSHW ;
  8
  9 : BFCLR                          ( -- )
 1Ø   PBLL LLEN ERASE
 11   <BFROT ;
 12
 13 : BFLN>                         ( -- )
 14   LNINS PBLL 2+ BOL 38 CMOVE
 15   CSHOW <BFROT EDMRK ;               -->
```

```
Screen: 127
  Ø ( valDOS file editor 1.Ø          )
  1
  2 : >BFNXT BFCPY DNCUR ;           ( -- )
  3
  4 : >BFLN BFCPY LNDEL ;            ( -- )
  5
  6 : BFRPL                          ( -- )
  7   CBLANK OOPSV
  8   PBLL 2+ BOL 38 CMOVE EDMRK
  9   <BFROT CSHOW EDMRK ;
 1Ø
 11 : TABSTP                         ( -- )
 12   XLOC 8 /MOD TABS +
 13   SWAP 2^ OVER C@ OR
 14   SWAP C! NORPT ;
 15                                     -->
```

```
Screen: 128
  Ø ( valDOS file editor 1.Ø          )
  1
  2 : TABCLR                         ( -- )
  3   XLOC 8 /MOD TABS +
  4   SWAP 2^ 255 XOR OVER
  5   C@ AND SWAP C! NORPT ;
  6
  7 : TAB CBLANK                     ( -- )
  8   38
  9   BEGIN
 1Ø     1- 1 AT XLOC +! XLOC
 11     38 MOD 8 /MOD TABS +
 12     C@ SWAP 2^ AND OVER Ø= OR
 13   UNTIL DROP XLOC
 14   38 /MOD SWAP TO XLOC
 15   IF DNCUR ENDIF CSHOW ;             -->
```

```
Screen: 129
  Ø ( valDOS file editor 1.Ø          )
  1
  2 : RUB                            ( -- )
  3   XLOC
  4   IF LFCUR Ø CURLOC C!
  5      CSHOW EDMRK
  6   ENDIF
  7   INSRT IF BYTDEL ENDIF NORPT ;
  8
  9 : BOTSCR                         ( -- )
 1Ø   15 XLOC - -DUP
 11   IF Ø DO DNCUR LOOP ENDIF ;
 12
 13 : OOPS                          ( -- )
 14   LNINS OOPSLN BOL 38 CMOVE
 15   EDMRK CSHOW NORPT ;                -->
```

```
Screen: 130
  0 ( valDOS file editor 1.0          )
  1
  2 : PTCHR                       ( -- )
  3   LSTCHR 13 = ?MULTI 1 <> AND
  4   IF
  5     1 TO PTFLG BONXT
  6   ELSE EDMRK
  7     INSRT IF BYTINS ENDIF
  8     LSTCHR >SCD CURLOC C!
  9     XLOC 37 =
 10     IF 1 TO PTFLG BONXT
 11     ELSE RTCUR ENDIF CSHOW
 12   ENDIF ;
 13
 14 : PRVSCR                     ( -- )
 15   #LNS 0 DO ROLLDN LOOP ;      -->
```

```
Screen: 131
  0 ( valDOS file editor 1.0          )
  1
  2 : NXTSCR                      ( -- )
  3   #LNS 0 DO ROLLUP LOOP ;
  4
  5 : SPLCHR                      ( -- )
  6   0 ?MULTI 0=
  7   IF 1+ 37 88 @ 25 + C! ENDIF
  8   TO ?MULTI ;
  9
 10 : MULTI                      ( -- )
 11   0 ?MULTI 0=
 12   IF 2+ 57 88 @ 25 + C! ENDIF
 13   TO ?MULTI ;
 14
 15                                -->
```

```
Screen: 132
  0 ( valDOS file editor 1.0          )
  1 : SPLT                        ( -- )
  2   CBLANK BOL EDWRK 38 CMOVE
  3   XLOC CLREOL 1 TO PTFLG BONXT
  4   CBLANK TO XLOC EDWRK XLOC +
  5   BOL XLOC + 38 XLOC - CMOVE
  6   EDMRK CSHOW ;
  7
  8 : -SPLT                       ( -- )
  9   CURLN TXTBOT <>
 10   IF CBLANK BOL EDWRK 38 CMOVE
 11     LNDEL UPCUR CBLANK OOPSV
 12     EDWRK XLOC + BOL XLOC +
 13     38 XLOC - CMOVE CSHOW EDMRK
 14     EDWRK OOPSLN XLOC CMOVE
 15   ENDIF NORPT ;                -->
```

```
Screen: 133
  0 ( valDOS file editor 1.0          )
  1
  2 FORTH DEFINITIONS
  3
  4 : .INFO EDITOR                ( -- )
  5   CR ." File start addr: "
  6   TXTBOT DUP U.
  7   CR ." File end addr: "
  8   TXTEND 2- DUP U.
  9   CR ." File count: "
 10   SWAP - U.
 11   CR ." Bytes free: "
 12   MEMTOP TXTEND -
 13   0 MAX U. CR ;
 14
 15 EDITOR DEFINITIONS             -->
```

```
Screen: 134
  0 ( valDOS file editor 1.0          )
  1
  2 : WIPEIT                      ( -- )
  3   88 @ 40 + 640 ERASE
  4   12 84 C! 0 752 C! CR ;
  5
  6 : FLFL                        ( -- )
  7   TXTBOT HOME ;
  8
  9 : FLLL                        ( -- )
 10   TXTEND 2- 15 0
 11   DO PRVLN LOOP HOME BOTSCR ;
 12
 13 : FLEN                        ( -- )
 14   CURLN NXTLN DUP
 15   2+ TO TXTEND 0 SWAP ! DC ; -->
```

```
Screen: 135
  0 ( valDOS file editor 1.0          )
  1 : SRCH
  2   SRCH$ C@
  3   IF CS CURLN XLOC + 1+ TXTEND
  4    OVER - SRCH$ COUNT MATCH SWAP
  5      IF CURLN XLOC + + DUP
  6        PRVLN DUP TO CURLN -
  7        TO XLOC 0 TO YLOC CURLN
  8        6 0 DO
  9          DUP PRVLN SWAP OVER
 10          <> AT YLOC +!
 11        LOOP TO DSPTOP GETLN DC
 12      ELSE DROP WIPEIT CR
 13        ." Not found" CRWT FLFL
 14      ENDIF
 15   ENDIF ;                      -->
```

```
Screen: 136
  Ø ( valDOS file editor 1.Ø        )
  1
  2 : GET$                  ( delm -- $ )
  3   >R PAD 2+
  4   BEGIN DUP C@ R =
  5   WHILE 1+ REPEAT DUP
  6   BEGIN DUP C@ DUP R <>
  7     SWAP Ø# AND
  8   WHILE 1+ REPEAT OVER - OVER
  9   1- C! 1- R> DROP ;
 1Ø
 11 : STSRCH                        ( -- )
 12   ." ? " PAD 31 EXPECT
 13   CR 88 @ 564 + SRCH$ 1+ CRAM
 14   1- SRCH$ C! SRCH$ 1+ DUP
 15   33 BSCD> SRCH ;                --->
```

```
Screen: 137
  Ø ( valDOS file editor 1.Ø        )
  1
  2 : INFL DOS                      ( -- )
  3   INSRT Ø TO INSRT
  4   BL GET$ (OPEN)
  5   IF DC LNINS
  6     BEGIN
  7       ?TERMINAL ?2BIG OR NOT DUP
  8       IF DROP DUP (RDB) ENDIF
  9     WHILE
 1Ø       DUP 155 =
 11       IF DROP 13 ENDIF
 12       TO LSTCHR PTCHR
 13     REPEAT (CLOSE) DROP
 14   ELSE CR
 15     ." Unable to load file"   --->
```

```
Screen: 138
  Ø ( valDOS file editor 1.Ø        )
  1     CRWT WIPEIT DC
  2   ENDIF TO INSRT ;
  3
  4 : FLST                          ( -- )
  5   CURLN TXTBOT 2DUP
  6   TXTEND CURLN - 1+ CMOVE
  7   - MINUS AT TXTEND +! FLFL ;
  8
  9 : TABRST                        ( -- )
 1Ø   TABS 4 68 FILL
 11   36 TABS 4 + C! DC ;
 12
 13 : BADSUB                        ( -- )
 14   CR ." Bad subcommand"
 15   CRWT WIPEIT DC ;               --->
```

```
Screen: 139
  Ø ( valDOS file editor 1.Ø        )
  1
  2 : SUBCMD                              ( -- )
  3   CS WIPEIT
  4   ." : " PAD 32 ERASE
  5   PAD 31 EXPECT CR PAD @
  6   SEL ( ST) 21587 -> FLST
  7       ( EN) 2ØØ37 -> FLEN
  8       ( FL) 19526 -> FLFL
  9       ( LL) 19532 -> FLLL
 1Ø       ( IF) 17993 -> INFL
 11       ( PS) 21328 -> STSRCH
 12       ( RT) 21586 -> TABRST
 13                  Ø -> DC
 14   NOSEL BADSUB
 15   SELEND ;                           --->
```

```
Screen: 14Ø
  Ø ( valDOS file editor 1.Ø        )
  1
  2 : EDTABT                        ( -- )
  3   CS WIPEIT ." Abort? "
  4   +Y/N 19 * DUP TO LSTCHR Ø=
  5   IF DC ELSE ?UPDAT Ø=
  6   IF CURLN TXTBOT - TO ED#IN
  7   ELSE TXTBOT DUP TO DSPTOP
  8   TO CURLN Ø TO XLOC Ø TO YLOC
  9   ENDIF ENDIF NORPT ;
 1Ø
 11 : EXIT  DOS                     ( -- )
 12   CS WIPEIT 8 84 C!
 13   .INFO CR ." Save '"
 14   EDN2$ COUNT TYPE ." ' ? "
 15   +Y/N 19 * DUP TO LSTCHR       --->
```

```
Screen: 141
  Ø ( valDOS file editor 1.Ø        )
  1   IF EDN2$ (OPEN) Ø=
  2     IF Ø DSKERR FLDNE =
  3       IF DROP EDN2$ (ENTER)
  4         DROP EDN2$ (OPEN)
  5       ENDIF
  6       Ø= IF DC ;S ENDIF
  7     ENDIF
  8     SWAP DROP TO EDFL#
  9     TXTBOT TXTEND 2- OVER -
 1Ø     EDFL# (WRITE) DROP EDFL#
 11     (ENDF) DROP EDFL# (CLOSE)
 12     CURLN TXTBOT - TO ED#IN
 13   ELSE
 14     DC
 15   ENDIF ;                        --->
```

```
Screen: 142
 Ø ( valDOS file editor 1.Ø          )
 1
 2 : CONTROL                    ( n -- )
 3   SEL
 4     19 -> EXIT     17 -> EDTABT
 5     28 -> UPCUR    29 -> DNCUR
 6     3Ø -> LFCUR    31 -> RTCUR
 7    126 -> RUB     127 -> TAB
 8      9 -> INTGL   155 -> BONXT
 9    255 -> BYTINS  254 -> BYTDEL
1Ø    157 -> LNINS   156 -> LNDEL
11     18 -> BFROT     2 -> <BFROT
12      3 -> BFCLR    11 -> >BFNXT
13     2Ø -> >BFLN     6 -> BFLN>
14     16 -> PRVSCR   14 -> NXTSCR
15     15 -> OOPS                 -->
```

```
Screen: 143
 Ø ( valDOS file editor 1.Ø          )
 1
 2     27 -> SPLCHR    8 -> CLREOL
 3     21 -> BFRPL    25 -> MULTI
 4     24 -> ROLLDN    5 -> ROLLUP
 5     22 -> SUBCMD   12 -> SRCH
 6     1Ø -> SPLT      7 -> -SPLT
 7    159 -> TABSTP  158 -> TABCLR
 8   NOSEL PTCHR SELEND ;
 9
1Ø : 2STROKE                    ( -- )
11   ?MULTI 1 =
12   IF DROP PTCHR
13   ELSE
14     ?MULTI 2 =
15     IF                         -->
```

```
Screen: 144
 Ø ( valDOS file editor 1.Ø          )
 1
 2     BEGIN
 3       8 53279 C! DUP CONTROL
 4       ?MULTI NOT ?TERMINAL OR
 5     UNTIL
 6     DROP ENDIF
 7   ENDIF
 8   Ø TO LSTCHR NORPT
 9   Ø 88 @ 25 + C! ;
1Ø
11 : EDTSTP DOS          ( $o $i -- )
12   DECIMAL 1 PFLAG !
13   PBL BLEN + LLEN + 2+
14   TO TXTBOT
15   EDN1$ (OPEN) ?SYSERR          -->
```

```
Screen: 145
 Ø ( valDOS file editor 1.Ø          )
 1
 2   SWAP DROP TO EDFL#
 3   Ø TXTBOT 2- ! 155 TXTBOT C!
 4   TXTBOT MEMTOP OVER - DUP
 5   <ROT EDFL# (READ)
 6   IF EDFL# (CLOSE)
 7     FLTBG CMDERR
 8   ENDIF
 9   DSKERR EOFERR = ?SYSERR
1Ø   EDFL# (CLOSE)
11   #UNTRN - DUP Ø= + TXTBOT +
12   DUP 2+ TO TXTEND Ø SWAP !
13   Ø TO ED#IN Ø TO ?UPDAT ;
14
15                                 -->
```

```
Screen: 146
 Ø ( valDOS file editor 1.Ø          )
 1
 2 : PROCESS DOS                 ( -- )
 3
 4   Ø GR. 1 752 C! CLS
 5   112 56Ø @ 6 + C!
 6   112 56Ø @ 23 + C!
 7   ."  File: " EDN2$ $.
 8   28 85 !
 9   ." #Bufs: " BLEN LLEN / .
1Ø   GETLN DC
11
12   PAD ?PADSM OVER TO ?PADSM =
13   PBL @ ?BUFSM = AND NOT
14   IF PBL BLEN ERASE ENDIF
15   BFSHW                          -->
```

```
Screen: 147
 Ø ( valDOS file editor 1.Ø          )
 1
 2   BEGIN
 3     INKEY$ DUP TO LSTCHR -DUP
 4     IF
 5       ?MULTI
 6       IF 2STROKE
 7       ELSE CONTROL ENDIF
 8     ELSE
 9       INSRT
1Ø       IF CBLANK ( PAUSE)
11          2DUP DROP DROP CSHOW
12       ENDIF
13     ENDIF
14     LSTCHR 19 =
15   UNTIL                          -->
```

```
Screen: 148
  Ø ( valDOS file editor 1.Ø        )
  1
  2   Ø 767 C! Ø 752 C!
  3   2 56Ø @ 6 + C!
  4   2 56Ø @ 23 + C!
  5   CLS CR
  6   ." Last edit in: " EDN2$
  7   $. CR .INFO CR
  8   PBL @ TO ?BUFSM
  9   DSKFLS ;
 1Ø
 11
 12
 13
 14
 15                                 -->
```

```
Screen: 149
  Ø ( valDOS file editor 1.Ø        )
  1
  2 FORTH DEFINITIONS
  3
  4 : EDIT DOS                  ( -- )
  5   GETARGS ?WRGARG
  6   TOPOM 4Ø - EDITOR TO MEMTOP
  7   DOS 44 ( ",") GETARG Ø=
  8   IF DUP ENDIF TO N1$ TO N2$
  9   Ø (CLOSE) N1$ (?OPEN)
 1Ø   IF
 11     Ø= FLOPN ?CMDERR
 12   ELSE
 13     DSKERR FLDNE = ?SYSERR
 14
 15                                 -->
```

```
Screen: 15Ø
  Ø ( valDOS file editor 1.Ø        )
  1
  2     CR N1$ $.
  3    ."  does not exist, create? "
  4     +Y/N NOT CR IF ;S ENDIF
  5     N1$ (ENTER) ?SYSERR
  6   ENDIF
  7   UNIT DIRTBL C@ ENTRY 1+ @
  8   N2$ (?OPEN)
  9   IF
 1Ø     Ø= FLOPN ?CMDERR
 11     UNIT DIRTBL C@ ENTRY 1+ @
 12   ELSE
 13     DSKERR FLDNE = ?SYSERR Ø
 14   ENDIF -
 15                                 -->
```

```
Screen: 151
  Ø ( valDOS file editor 1.Ø        )
  1
  2   UNIT FSMGET
  3   FSMAP 3 + @ SWAP - DUP 2Ø <
  4   IF CR
  5     ." Warning, disk space " Ø>
  6     IF ." low"
  7     ELSE ." empty" ENDIF
  8     ." , edit?"
  9     +Y/N NOT CR IF ;S ENDIF
 1Ø   ELSE DROP ENDIF
 11   UNIT DIRTBL C@ ENTRY C@ 32 AND
 12   IF CR
 13    ." File is locked, edit? "
 14    +Y/N NOT CR IF ;S ENDIF
 15   ENDIF                         -->
```

```
Screen: 152
  Ø ( valDOS file editor 1.Ø        )
  1
  2   N1$ EDITOR EDN1$ OVER
  3   C@ 1+ 16 MIN CMOVE
  4   DOS N2$ EDITOR  EDN2$ OVER
  5   C@ 1+ 16 MIN CMOVE
  6   EDTSTP Ø TO INSRT
  7   TXTBOT DUP TO DSPTOP TO CURLN
  8   Ø TO XLOC Ø TO YLOC GETLN
  9   CR .INFO CRWT PROCESS ;
 1Ø
 11 FORTH
 12
 13
 14
 15                                 -->
```

```
Screen: 153
  Ø ( valDOS file editor 1.Ø        )
  1
  2 : LL EDITOR                 ( -- )
  3   EDN2$ EDN1$ 16 CMOVE
  4   PAD ?PADSM <>
  5   IF Ø TO XLOC Ø TO YLOC PBL
  6   BLEN + LLEN + 2+ DUP TO TXTBOT
  7   DUP TO DSPTOP TO CURLN
  8   GETLN ENDIF EDTSTP PROCESS ;
  9
 1Ø : WHERE DOS                 ( -- )
 11   FLFL# FLBUF@ -DUP Ø=
 12   IF
 13     CR ." No error on record..."
 14     CR QUIT
 15   ENDIF                         -->
```

```
Screen: 154
  Ø ( valDOS file editor 1.Ø          )
  1
  2   Ø (CLOSE) 14 + Ø
  3   FLNME EDITOR EDN1$ 16 CMOVE
  4   DOS FLNME EDITOR EDN2$
  5   16 CMOVE EDTSTP
  6   1- TXTBOT + DUP PRVLN
  7   SWAP OVER - 1- TO XLOC
  8   DUP TO CURLN Ø TO YLOC 6 Ø
  9   DO
 1Ø     DUP PRVLN
 11     SWAP OVER <> AT YLOC +!
 12  LOOP
 13  TO DSPTOP
 14  1 TO INSRT
 15  PROCESS ;                    -->
```

```
Screen: 155
  Ø ( valDOS file editor 1.Ø          )
  1
  2 : #BUFS EDITOR             ( n -- )
  3   5 MAX 32Ø MIN LLEN *
  4   ' BLEN ! Ø TO ?PADSM
  5   PBL BLEN + LLEN + 2+
  6   DUP TO TXTBOT DUP
  7   TO DSPTOP TO CURLN
  8   Ø TO XLOC Ø TO YLOC ;
  9
 1Ø FORTH
 11
 12 BASE !
 13
 14
 15
```

```
Screen: 156
  Ø
  1
  2
  3
  4
  5
  6
  7
  8
  9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen: 157
  Ø
  1
  2
  3
  4
  5
  6
  7
  8
  9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen: 158
  Ø
  1
  2
  3
  4
  5
  6
  7
  8
  9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen: 159
  Ø
  1
  2
  3
  4
  5
  6
  7
  8
  9
 1Ø
 11
 12
 13
 14
 15
```

```
Screen: 160
  0 ( file editor load message      )
  1
  2 CLS
  3 CR CR CR
  4 ." valDOS and the basic DOS" CR
  5 ." commands must be loaded"  CR
  6 ." before the file editor"   CR
  7 ." is compiled."             CR
  8 CR
  9 ." Insert the valDOS I disk" CR
 10 ." and load the necessary"   CR
 11 ." routines."                CR
 12 CR CR
 13 FLUSH
 14
 15
```

```
Screen: 161
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 162
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 163
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 164
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 165
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

```
Screen: 166
 Ø
 1
 2
 3
 4
 5
 6
  6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 167
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 168
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 169
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 17Ø
 Ø CONTENTS OF THIS DISK:
 1
 2 CASE STATEMENTS:              5Ø LOAD
 3 SCREEN CODE CONVERSION:       6Ø LOAD
 4 KEYSTROKE WORDS:              7Ø LOAD
 5 DOS FILE EDITOR 1.Ø:         1ØØ LOAD
 6
 7  **** CAUTION **** CAUTION ****
 8
 9  This is a DOS format disk
10  with screens 5Ø-79 and 1ØØ-179
11  locked out for FORTH source
12  code. Do not store FORTH
13  code on screens that are not
14  locked out!
15
```

```
Screen: 171
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 172
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 173
 Ø
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 174
 Ø ( valDOS error messages                )
 1 Illegal filename
 2 Bad/Mismatched unit(s)
 3 Bad free space map
 4 File already exists
 5 Directory is full
 6 Disk is full
 7 Filename is ambiguous
 8 File does not exist
 9 No room for buffer
10 End of file encountered
11 File is not open
12 Illegal file number
13 File is locked
14 Bad argument list
15 File is open
```

```
Screen: 175
 Ø File is too big
 1 File is random
 2 File is not random
 3 No room for random map
 4 Random map is bad
 5 File is a device file
 6 File is not a device file
 7 Illegal access to device file
 8
 9
10
11
12
13
14
15
```

```
Screen: 176
 Ø ( Error messages                       )
 1
 2 Stack empty
 3
 4 Dictionary full
 5
 6 Wrong addressing mode
 7
 8 Is not unique
 9
10 Value error
11
12 Disk address error
13
14 Stack full
15
```

```
Screen: 177
 Ø Disk Error!
 1
 2 Dictionary too big
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
Screen: 178
  0 ( Error messages                    )
  1
  2 Use only in Definitions
  3
  4 Execution only
  5
  6 Conditionals not paired
  7
  8 Definition not finished
  9
 10 In protected dictionary
 11
 12 Use only when loading
 13
 14 Off current screen
 15


Screen: 179
  0 Declare VOCABULARY
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
```

HANDY REFERENCE CARD
valFORTH™ SOFTWARE SYSTEM
valDOS

Command Words

CLOSE Release file buffer and update file.
CLOSE (filenum)

COPY Transfer one or more file to another file.
COPY outfile(/A)=infile1(,infile2(,...))
/A = append file(s) to outfile.

DIR Display list of files on disk.
DIR (filespec)
DIR unit

DISKCOPYn Copy an existing diskette to another diskette.
DISKCOPY1 single-drive copy
DISKCOPY2 multi-drive copy

EDIT Edit a (FORTH) source file.
EDIT infile(,outfile)

ENDFIL Mark the current position within an open file as the end of that file.
ENDFIL filenum

EOF Position the file cursor at the end of the file.
EOF filenum

ENTER Enter a filename into a disk directory.
ENTER filename

FDUMP Perform a hex/ASCII dump of a file, 8 bytes/line.
FDUMP filename(/W)
/W = dump in 16 bytes/line format.

FILE-IT Transform screen format to DOS file format.
FILE-IT 1stscrn,lastscrn,filename

FLOAD Compile a FORTH source file from disk.
FLOAD filename(/C)
FLOAD filename(,linenum)
/C = continue load from last line edited.

FMOVE Single drive interdisk file transfer
FMOVE outfile=infile

FORMAT Format a diskette for use.
FORMAT (unit)

FSPACE Move through a file relative to the current file position.
FSPACE filenum,count

KILL Remove a file from the directory.
KILL filespec1(/N)(,filespec2(/N)(,...))
/N = no verification message on wild-card names.

LOCK Write and modify protect files on disk
LOCK filespec1(/N)(,filespec2(/N)(,...))
/N = no verification message on wild-card names.

NAMEDISK Name a diskette.
NAMEDISK (unit)

OPEN Open a file for access.
OPEN filename

OPEN? List all files currently open.
OPEN?

PRINT Display a text file on the current output device.
PRINT filename (/N)(,linenum)
/N = no line numbers.

READ Read a file into memory.
READ filename,address(,count)
READ filenum,address(,count)

RENAME Rename a file.
RENAME newname=oldname

REWIND Reposition the file cursor to the beginning of the file.
REWIND filenum.

SETUNIT Set the default unit.
SETUNIT unit

UNLOCK Unprotect a file so that it may be modified.
UNLOCK filespec1(/N)(,filespec2(/N)(,...))
/N = no verification message on wild-card names.

WRITE Write an area of memory to a file.
WRITE filename,address,count
WRITE filenum,address,count

System Words

(?OPEN) $ --- (f 1)/0
This command checks to see if the specified file is currently open. If an error occurs, a zero is returned, otherwise a flag and a one are returned. If the flag is one, the file is open, otherwise it is closed.

(CLOSE) fl# ---
The file whose file access number is "fl#" is closed for access. If "fl#" is 0, all open files are closed.

(ENDF) fl# --- f
The current cursor position within the file is made the end of the file. All data past the cursor is lost, and disk space reclaimed. A one indicates that no errors occurred.

(ENTER) $ --- f
The filename on the top of stack is entered into the directory on the unit specified within the filename. If no unit is given, the default unit is assumed. A one is returned if there was no error, else zero.

(KILL) $ --- f
The specified filename is removed from the directory on the unit specified within the filename. A one is returned if there was no error, else zero.

(LOCK) $ --- f
The specified file is locked, thus preventing any modification to that file. A one is returned if there was no error, else zero.

(OPEN) $ --- (addr fl# 1)/0
Opens the specified file. If an error occurs, a zero is returned, otherwise the buffer address, the file access number, and one are returned.

(RDB) fl# --- (b 1)/0
The next byte of the file whose access number is "fl#" is read. If there was no error, the byte is returned along with a one, otherwise only a zero is returned.

(READ) addr count fl# --- f
The next "count" bytes of the file whose file access number is "fl#" are written to the block of memory specified by "addr". A one is returned if all went well, otherwise a zero is returned.

(REN) $n $o --- f
The file whose name is "$o" is renamed to "$n". A one is returned if no error occurred.

(SPACE) cnt fl# --- f
The file cursor of the specified file is moved relative to its current position by the signed value specified by "cnt". A one is returned if there was no error, otherwise zero is returned.

(UNLOCK) $ --- f
The specified file is unlocked, allowing the file to be modified or killed. A one is returned if there was no error, else zero.

(WIND) n fl# --- f
If n is one, the file whose access number is "fl#" is rewound so that it may be completely re-read. If n is zero, the file cursor is moved to the very end of the file. A zero is returned if there was an error, otherwise a one is returned.

(WRB) b fl# --- f
The byte "b" is written to the file whose access number is "fl#". A one is returned if there was no error, otherwise a zero is returned.

(WRITE) addr count fl# --- f
The block of memory "count" bytes long at the specified address is written to the file whose file access number is "fl#". A one is returned if all went well, otherwise a zero is returned.

CHKDIR --- n
Returns the number of occurrences that the last filename converted by FNCON (below) appears in the directory of the unit specified in the filename.

FLBUF@ fl# --- addr
Returns the address of the 16 byte information block of the file whose access number is "fl#". This block contains the following information:

addr+0:	File status byte
addr+1:	Current size of file. High bit set indicates that the file is updated.
addr+3:	First (DOS) sector of file.
addr+5:	(FORTH) sector currently in file buffer.
addr+7:	Number of bytes into current sector.
addr+8:	Unit associated with the file.
addr+9:	Entry number in the directory.
addr+10:	Non-zero = current sector is updated.
addr+11:	(reserved for) Current random block
addr+13:	(reserved for) Random block updated?
addr+14:	Number bytes into the file. (2 bytes)
addr-128:	Address of file buffer

FNCON ($ -- f)
Convert a filename to directory format. Returns true if the filename is legal, else zero.

FNFLD --- addr
Returns the address of the eleven byte formatted filename produced by FNCON.

System QUANS

#ENTRIES --- n
Returns the number of entries that matched the filename last checked by CHKDIR. CHKDIR returns this value automatically.

#FREE --- n
Returns the number of free directory entries in the directory last checked by CHKDIR (below).

#UNTRN --- n
Returns the number of bytes left untransferred in the last read or write operation. Usually returns zero unless an error or eof occurred.

?WILD --- f
Returns a one if any wild cards appeared in the last filename converted by FNCON, otherwise it returns a zero.

DFLUNT --- n
Returns the current default unit.
0 = Atari DOS drive 1.

DSKERR --- n
Returns the error number of the last reported DOS error. See list of error names.

FNSWCH --- c
Returns the ASCII code for the switch "/c" found in the last filename converted by FNCON. If no switch was found, this returns zero.

UNIT --- unit
Returns the unit number specified by the last filename sent to FNCON. If no filename was present in the specification, the default unit is returned. Note that DOS drive 1 is FORTH drive 0. UNIT returns the FORTH drive value.

System Errors (constants)

AMBNME --- n
Ambiguous filename (no wild cards allowed).

BADFL# --- n
Bad file access number.

BADNME --- n
Bad/illegal filename.

BADUNT --- n
Bad unit specification.

BADFSM --- n
Bad free space map.

DIRFUL --- n
Directory full.

DSKFUL --- n
Disk is full.

EOFERR --- n
End of file reached.

FLDNE --- n
File does not exist.

FLEXST --- n
File already exists.

FLNOPN --- n
File is not open.

FLOPN --- n
File is open.

FLTBG --- n
File is too big.

FLWRPT --- n
File is write protected (locked).

TMFOPN --- n
Too many files open.

WRGARG --- n
Wrong/no arguments given.

HANDY REFERENCE CARD

valFORTH™ SOFTWARE SYSTEM

valDOS

File Editor Command Summary

Below is a quick reference list of all the commands which the video editor recognizes.

Entering the Edit Mode

(executed outside of the edit mode)

EDIT infile(,outfile) (---)
Enter the edit mode and edit "infile". If "outfile" is specified, send edited text to "outfile", otherwise send it to "infile".

LL (---)
Re-edit the last file edited.

WHERE (---)
Enter the edit mode and position the cursor over the word that caused a compilation error.

#BUFS (#lines ---)
Sets the length (in lines) of the storage buffer. The default is five.

Cursor Movement

(issued within the edit mode)

ctrl ↑ Move cursor up one line, scrolling the file down one line if necessary.
ctrl ↓ Move cursor down one line, scrolling the file up one line if necessary.
ctrl ← Move cursor left one character, wrapping to the right edge if moved off the left.
ctrl → Move cursor right one character, wrapping to the left edge if moved off the right.
RETURN Position the cursor at the beginning of the next line. Insert line if at the end of the file.
TAB Advance to next tabular column.
ctrl TAB Clear tab stop at current cursor location.
shift TAB Set tab stop at current cursor location.

Editing Commands

(issued within the edit mode)

ctrl INS Insert one blank at cursor location, losing the last character on the line.
ctrl DEL Delete character under cursor, closing the line.
shift INS Insert blank line above current line.
shift DEL Delete current cursor line, closing the file.
ctrl M Insert blank line below current line.
ctrl I Toggle insert-mode/replace-mode. (See full description of ctrl-I).
BACKS Delete last character typed, if on the same line as the cursor.
ctrl H Erase to end of line (Hack).
ctrl V Enter the subcommand mode. (see below)

Buffer Management

(issued within the edit mode)

ctrl T Delete current cursor line sending it to the edit buffer for later use.
ctrl F Take the current buffer line and insert it above the current cursor line.
ctrl K Copy current cursor line sending it to the edit buffer for later use.
ctrl U Take the current buffer line and copy it to the current cursor line.
ctrl R Roll the buffer making the next buffer line current.
ctrl B Roll the buffer backwards making the previous buffer line on the screen current.
ctrl C Clear the current buffer line and performs a ctrl-B.

Note: The current buffer line is last line visible on the video display.

Scrolling/Saving

(issued within the edit mode)

ctrl X Scroll the edit window up a line within the file.
ctrl E Scroll the edit window down a line within the file.
ctrl P Scroll the edit window up 16 lines within the file.
ctrl N Scroll the edit window down 16 lines within the file.
ctrl S Save the changes made to the current file and exit the edit mode.
ctrl Q Quit the edit session forgetting all changes made to the current file.

Special Keys

(issued within the edit mode)

ESC Do not interpret the next key typed as any of the commands above. Send it directly to the screen instead.
ctrl J Split the current line into two lines at the point where the cursor is.
ctrl G Splice (unsplit) the current line with the line above it.
ctrl O Corrects any major editing blunders.
ctrl L Continue searching for the pattern entered in the subcommand mode.
ctrl Y Enter the repeat mode. The next command or character typed will be repeated until a stop condition is met, or until a console key is pressed. Used mostly with ctrl-P, ctrl-N, and the cursor commands.

Subcommands

(entered in the subcommand mode)

ST <return> Make the current line the start of the file. (i.e., hack off the beginning)
EN <return> Make the current line the end of the file. (i.e., hack off the end)
FL <return> Position the cursor on the first line of the file.
LL <return> Position the cursor on the last line of the file.
RT <return> Reset the TAB stops to their original settings.
PS <return> Enter the pattern search submode. The user will be prompted to enter the search string. The ctrl-L command will continue the search.
IF filename <return>
Insert the specified file into the file just after the current cursor line. This is useful for pulling subroutines from another file. This can be stopped at any time by pressing a console key.